The 27th annual workshop of the Swedish Artificial Intelligence Society (SAIS),
14–15 May 2012, Örebro, Sweden
Published by Linköping University Electronic Press
http://www.ep.liu.se/ecp_home/index.en.aspx?issue=071

Interleaving Configuration Planning and Action Planning in Robotic Ecologies

Lia Susana d.C. Silva-Lopez, Lars Karlsson
{lia.silva,lars.karlsson}@aass.oru.se
School of Science and Technology
Örebro University

Abstract

In the context robotic ecologies, the Configuration Planning Problem (CPP) is concerned with ways to create a flow of information and a flow of causality between the members of an ecology and their environment, in such a way that a goal is satisfied. Ways to solve this type of CPP have been devised and we discuss some in this paper. However, most consider only the causal or information aspects of the members of an ecology, or rely on abstract actions and task hierarchies. A problem with considering only causal or information aspects, is that it complicates modelling situations when both occur and relate to each other. A problem with abstract actions and task hierarchies is that it can lead to incompleteness, and require effort and skill to write down. The goal of this paper is to discuss a way to interleave configuration planning with action planning, in which direct interconnections of either causal or information links, are used to solve the problem of building configurations of networked robotic systems. We show with examples how this approach works, discuss some future directions on where we want this work to get and more specifically in the context of the Giraff+ system.

1 Introduction

Robotic ecologies, as proposed by Saffioti and Broxvall in [10], are collections of devices and programs with cognition and communication capabilities, in which the notion of robot emerges from the interaction between the elements of the ecology. In robotic ecologies, the purpose is to use a synergy of small devices to fulfil goals in a flexible way, instead of building complex universal robots. In the Configuration Planning Problem (CPP) for the context of robotic ecologies (See section 2 of this paper), the purpose is to satisfy a goal by interconnecting a set of available components with constraints between them. A configuration here, can be seen as a collection of functional elements that contribute with their capabilities to fulfil a goal, connected to each other by causal links and information channels.

In this paper, we present some preliminary explorations of how to integrate two kinds of interactions between entities in a robot ecology. We consider information flows, for instance from a light sensor to a process that tests whether a certain threshold of light has been exceeded to an actuator for adjusting the blinds of a window. We also consider causal flows, e.g. in order to obtain information using a video camera, the light first has to be switched on. An important difference between these two types of interactions is that information flows can never interfere with each other, whereas causal flows can interfere by changing the state of the environment. For instance, assume that the system also has the goal to keep the room dark because a person is sleeping there. Switching on the lights in order to use the video camera would interfere with that goal. In addition, for the purpose of this paper we consider causality to apply from one time step to a later one, whereas information flows between entities that are active at the same time step. Our work is motivated by the fact that in the current literature for solving the CPP, information inputs and outputs are either treated as preconditions and postconditions [3] or are not considered in the action planning. Moreover, goals are defined as actions to perform, but there is no mechanism for defining information as a goal.

This work is a step towards building a system capable of generating configurations for multiple goals set by an activity recognition system, of dynamically estimat-

ing preferences according to different criteria, and of updating state variables as the configurations executes, in a real world scenario. Our system is intended to be a part of the **Giraff+ system** [1], an adaptive system consisting of a Giraff robot, a sensor network, an activity recognition system [7](also called context recognition in Figure 1), and a configuration planner. The purpose of Giraff+ is to help improving quality of independence of the elderly, by providing personalised services of long-term monitoring in their homes and easier social interaction. Examples of services for improving quality of independence, that Giraff+ can provide in a home would be:

- Displaying time-line information of events of interest in combination with other contextual information, such as blood pressure readings after starting the use of a new medication or blood sugar measurements during days where the patient exercises.
- Reminding and assisting the person in important events, such as taking medication or taking blood sugar or heartbeat measurements.
- Providing long term statistics on e.g. amounts of physical activity or sleep.

Giraff+ is going to be tested in more than a dozen real homes, therefore it should be suitable for real application domains.

There are a number of ways in which a person can interact with its surroundings, and a number of ways to associate sensed data to the activities of a person. The role of the activity recognition system is to infer which activities is the human performing, based on information obtained from the sensor network. For more on activity recognition, see Ullberg et al [12], in which some approaches for recognizing human activities are discussed, and an algorithm for enabling long term and continuous activity recognition based on a temporal reasoner is proposed. For enabling activity recognition, raw sensor readings may not be enough to get enough information. There will be cases where the information may not be available directly, and an interaction with the environment may be needed to get it. The role of the configuration planner is to configure the network, according to the changing nature of the environment and the needs of the activity recognition system, in order to observe or change the value of a state variable.

To illustrate the Giraff+ system we will use Figure 1, where an environment is abstracted as state variables

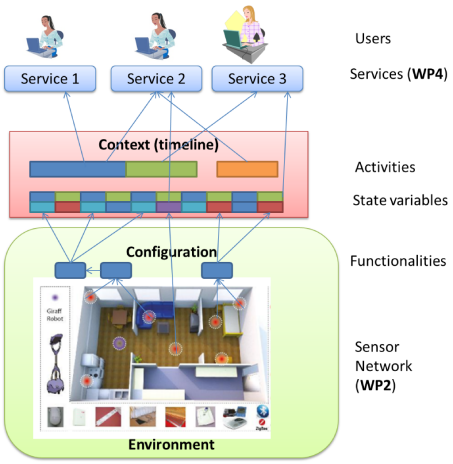

Figure 1: Functional Overview of the Giraff+ System.

that have a value at each point of time. These state variables may represent, for instance, aspects of the layout of the apartment (rooms, furniture), position and motion of individuals in the apartment (e.g. person1 in bedroom, on bed, not moving) or of items in the apartment. State variables may be observable through sensors. But in some cases, to observe a state variable, other state variables need to be changed e.g. to obtain the heart beat rate of a person, the lights of the room where the person is have to be turned on. Also, transformations on raw sensor data may be needed to obtain the value of a state variable e.g. the raw RSSI values between some bluetooth devices needs be transformed into the position of a person, but they can also be transformed into the trajectory of a movement, or transformed into a command for opening a door. Now, a functionality is a program that either operates directly on a sensor or actuator, or processes information from and/or delivers information to other functionalities. Note that a sensor or actuator can perform several functionalities, depending on which programs make use of it. The configuration planner builds configurations that make the sensor network provide the information (state variable values) needed by the activity recognition system, and may also change state variables.

This paper is organised as follows: Section 2 reviews related work, Section 3 explains the conceptual framework for our approach on configuration and action planning, Section 4 presents explanatory examples, and Section 5 describes ideas for future work.

2 Related Work

The CPP and similar problems have been studied in a number of fields, from which Robotic Ecologies and Web Service Composition can be pointed out.

Robotic ecologies, and more particularly ecologies of networked Physically Embedded Intelligent Systems (PEIS) as proposed by Saffioti and Broxvall [10], study how the overall functionalities of robotic systems go beyond the functionality of isolated robots, by adding communication and cognition capabilities into the robots. The CPP in robotic ecologies studies ways to interconnect the members of an ecology, in order to reach a goal.

Lundh et al in [6], propose an approach to automatically generate configurations for a given task in a given environment. An action planner defines which actions need to be accomplished to get to the goal, and for each action, a configuration is made. Methods similar to Hierarchical Task Networks (HTNs) planning were used to hold information on how each action should decompose into configurations. Terminating functionalities were used to determine when a configuration completed its task. Goals were defined in terms of actions. The capabilities of the members of an ecology were modelled as programs that can interact with the environment through sensing or actuating, and/or use information to produce additional information; such programs are called functionalities. Functionalities have causal preconditions, causal postconditions, information inputs and outputs, and also have costs and resources. Preconditions specify under which circumstances the functionality can be used, postconditions indicate how the functionality transforms the world state after its execution, information inputs indicate which information is required for the functionality to execute, and information outputs indicate which information is produced during functionality execution. A different approach is suggested by Gritti in [2], were a reactive configuration algorithm reconfigures the ecology with the available components, in the event of a failure. This approach is appropriate for very dynamic scenarios, because it inherently removes from the search space those functionalities that are not available in the time when they are needed.

Web service composition is concerned with ways of interconnecting self-contained, self-describing, modular applications that can be published, located, and invoked across the web. A composite service is a set of services and the control and data flow among them [3]. A number of approaches have been explored, from genetic algorithms, to neo-classical planning. For example, Tang et al [11] propose a solution to the optimal web service selection problem; this problem deals with selecting web services so that the composite web service give the best overall performance, and it can be seen as the problem of finding an optimal solution to the CPP under similar parameters. The authors propose a genetic algorithm with mutation and knowledge-based crossover as operators, in which each individual is a plan for Web Service Selection; a local optimizer improves individuals in the population, while the crossover operator evaluates constraints between services. In contrast, Peer [8] proposes an approach in which the problem of web service composition is automatically converted into an AI planning problem (see [9]), represented in *PDDL*; after that, the goal and domain description are matched against available AI planners, in order to solve the task.

Our approach to configuration planning is closer to the field of robotic ecologies, and builds on the one of Lundh et al, but unlike them we don't It does not rely on abstract actions or task hierarchies. Instead, we make direct connections between the different components needed to build the configuration, depending on the type of links needed. We do this because, while task hierarchies can speed up planning [9], they can also lead to incompleteness, and require effort and skill to write down, especially every time we add more functionalities into the system.

3 States and Functionalities

The ideas proposed in this paper are built upon the theoretical background presented in [5]. In this way, for the context of this paper, a configuration is a set of elements called functionalities, and the connections between them. Every time a functionality has an effect that changes properties of the world, the state of the world is changed.

3.1 State

A state s holds information that describes the world, in the form of a possible assignment of values to state variables. This assignments can be properties of certain objects (e.g. position(light1) = kitchen, near(light1,stove) = true, near(light1,fridge) = true) or the current state of

the different devices on our system (e.g. light1 = on, phone-bt = off).

Time is not explicitly contemplated as part of a state.

3.2 Functionalities

A functionality is a program that interacts with other programs by exchanging data, and/or with the physical world by effecting or observing the value of various properties of entities. These properties are described as state variables. Functionalities can operate on devices, sense, actuate, and may use information to generate information. They can execute simple actions and generate simple information pieces, or execute more complicated tasks and generate more information. For example, we can have a functionality that provides the location of an object as an output and requires no inputs, or we can have many functionalities together in a configuration and give the same output. In this work, we will use the definition by Lundh [5], in Equation 1.

$$f = \langle Id, I, O, \phi, Pr, Po, Re, \text{Cost} \rangle \quad (1)$$

Every functionality has an Id, a set of inputs I and a set of outputs O that can be of different types, causal preconditions Pr that indicate which values of state variables should hold before the execution of the functionality, causal postconditions Po that state which values of state variables will hold after the execution of the functionality, a transfer function ϕ that represents a transformation from inputs to outputs of the information before execution, to the information after execution, and some resources Re and costs $Cost$. At this early stage of our work, we have no particular definition for costs and resources, but they will receive a great deal of attention later on.

To refer to the element Id of a functionality f, we will write Id_f, and in a similar way we will write Pr_f,Po_f,I_f, and O_f.

4 Connections and Configurations

A connection between functionalities can be formed either by an information link, or by a causal link. Connections are formed when one functionality satisfies an information input or a causal precondition with an information output or a causal postcondition. Configurations are formed by building connections between functionalities. Figure 2 shows an example of a configuration.

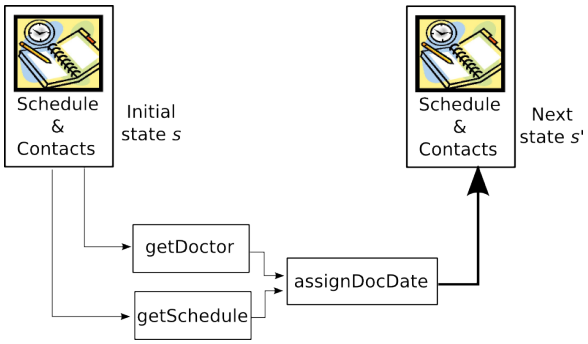

Figure 2: Example of a Configuration. Thin lines are information links. Thick lines are causal links.

4.1 Connections

A connection states a relationship between two functionalities, that can either be defined in terms of an information link, or a causal link.

Information links state that the output k of a functionality a (O_a^k) satisfy the input j of functionality b (I_b^j). Causal links state that the effects k of functionality a (Po_a^k) into the world, satisfy condition j that functionality b requires for execution (Pr_b^j).

The type of link also states the relative execution time of the functionalities in each connection: when two functionalities are related by an information link, they execute within the same interval of time (which we will call time-step), but when they are related by a causal link, the functionality that provides the postcondition for the link executes in a time-step which precedes the one of the functionality that requires such condition. Please note that with time-step we mean a finite interval of time, not a single time point.

In this work, we will use the definition of connections in Equations 2 and 3.

$$Cxi = \langle destF, sourceF, \text{infotype}, rT \rangle \quad (2)$$

$$Cxc = \langle destF, sourceF, Sv, SvV, rTd, rTs \rangle \quad (3)$$

Every connection contains a destination functionality $destF$, a source functionality $sourceF$, and a relative execution time rT. The relative execution time in the connection, is used to represent orderings between execution times of the functionalities. Information links contain the type of information required *infotype*, in which the name of the state variable can be used to describe

the piece of information that relates the functionalities in the link. Causal links contain a state variable assignment, where *Sv* is the name and *SvV* is the value of the state variable.

For example, suppose we have a causal goal for the system, of setting a slot of our free time with a Doctor's appointment; we have our schedule and our contact list in the form of state variables, functionalities $Id_f = getDoctor$, $Id_f = getSchedule$, and $Id_f = assignDocDate$. Functionality $Id_f = getDoctor$ knows which number is associated to the Doctor and has it as an output. Functionality $Id_f = getSchedule$, gets which slots are available in our schedule and has them as an output. Functionality $Id_f = assignDocDate$, negotiates with the Doctor's appointment system and, as a postcondition, it sets a slot of our free time with the value of a Doctor's, given a number associated to our Doctor and our available times as inputs.

In the case of building a configuration for the given goal by using this elements, we will first have a causal connection between $Id_f = assignDocDate$ as *sourceF*, and the goal as *destF*. In this connection, which *Sv* and *SvV* is a condition of setting a field of the schedule with the value of a doctors appointment, and the execution time of *SourceF*, happens before executing *destF*. In the special case of a goal as a destination functionality, the goal is artificially treated as another functionality; this means that to have the effect we want in the world, this functionality would need to be executed first. We may also have to perform several information connections, for example, connecting $Id_f = getDoctor$ to $Id_f = assignDocDate$ and $Id_f = getSchedule$ to $Id_f = assignDocDate$; in this case, *infotype* would be the piece of information provided in each connection, which can be $phone-nr(Doctor)$ for $Id_f = getDoctor$ to $Id_f = assignDocDate$ and $schedule(freeSpaces)$ for $Id_f = getSchedule$ to $Id_f = assignDocDate$, having all of them a simultaneous execution, since $Id_f = assignDocDate$ needed both pieces of information to execute.

4.2 Configurations

A configuration is a set of connections between a subset F of all functionalities in the world $\boldsymbol{F}$, that satisfy a Goal. In this work, we will use the definition of configuration in Equation 4.

$$c = \langle F, ICX, CCX \rangle \tag{4}$$

Here, F is the set of functionalities used in the configuration, *ICX* is the set of information links, and *CCX* is the set of causal links in the configuration. Relative execution times of functionalities are contained inside each connection.

4.2.1 Admissibility

For a configuration to be admissible, it must not violate constraints on costs, resources, information links and causal links. A proper way to evaluate admissibility on costs and resources will be devised later on in our work, but for the approach presented in this paper, we checked for information admissibility and causal admissibility.

To satisfy information admissibility in a configuration, all inputs in each functionality instance should be connected to the output of another functionality. To satisfy causal admissibility, all Preconditions in a functionality should be satisfied before the execution of the functionality, and no Postconditions of the functionalities in the configuration should be in conflict with the causal links of the configuration. The set of links should be acyclic.

When both information and causal admissibility are satisfied in a configuration, we can say that such configuration is admissible, and consider it a candidate configuration for the goal.

4.2.2 Goals

Since the definition of functionality implies both transforming information and interacting with the physical world, then is reasonable for the goal of a configuration to reflect this. We defined two types of goals: information goals and action goals, in Equations 5 and 6.

$$g_i = \langle Stv \rangle \tag{5}$$

$$g_a = \langle Stv, StvV \rangle \tag{6}$$

Goals are defined in terms of state variables. *Stv* is a state variable. If the goal is to obtain (or sense) the value of *Stv*, then it is an information goal. For information goals, the outputs of functionalities are examined to find which one delivers the value of *Stv*. If the goal is to interact with the world in such a way that we set the value of state variable *Stv* to *StvV*, then it is an action goal. For action goals, the postconditions of functionalities are examined to find an effect in which the value of *Stv* is set to *StvV*.

For the planning in the algorithms in Section 5, the Goal can be seen as a Goal Functionality, in which the inputs of the functionality are the information goals, and the preconditions are the action goals.

4.3 Configuration Planning Problem

In our Configuration Planning Problem, given a goal g_i or a goal g_a, a set of available functionalities $\boldsymbol{F}$ and a set of state variable assignments in the start state s, find an *admissible* configuration c such that $F_c \subseteq \boldsymbol{F}$, in such a way that the goal is satisfied.

5 An Algorithm for Configuration Planning

Our first approach to generating configurations with both direct causal and information links between the functionalities, is presented in Algorithms 1 and 2. We have an implementation of this approach in C++, available for downloading in [4].

Algorithm 1 (*ConAc*) requires the assignments of state variables in the start state s, a list of all functionalities in the world **F**, and a goal g, and delivers a list L of all admissible configurations. L is a global variable for algorithms 1 and 2. *ConAc* builds a functionality $f_g \in \mathbf{F}$, in which the inputs of the functionality are the information goals of g, and the preconditions are the action goals of g. Functionality f_g is added to (currently empty) partial configuration c, and into the list of functionalities of c, F_c, and an initial state functionality (f_i) is used to emulate the conditions of s in order to be able to establish causal links to it, but no causal or information connections have been performed so far, so ICX_c and CCX_c are empty in this assignment. Then, *ConAc* calls *genConf* with the goal functionality as the only member of the partial configuration. Then, *genConf* recursively aggregates all admissible configurations into L. In the end, *ConAc* returns the list L of all admissible configurations suitable for goal g.

Algorithm 2 (*genConf*), requires a partial configuration c, a list of all functionalities in the world **F**, and the relative time of the last source functionality added into the configuration RT. *genConf* modifies the list of all admissible configurations L by adding admissible configurations as it searches partial configurations. Please note that after execution of Algorithm Algorithm 1, **F** contains f_g and f_i.

Algorithm 1 ConAc. State variables in the start state s, available functionalities **F**, goal g, list of all admissible functionalities L

Require: s,**F**,g
Ensure: L
1: build goal functionality $f_g \in \mathbf{F}$ using goal g
2: build initial functionality $f_i \in \mathbf{F}$ using s
3: $c \leftarrow \langle \{f_g, f_i\}, \emptyset, \emptyset \rangle$
4: $L \leftarrow \emptyset$
5: $genConf(c,\mathbf{F},0)$ */ Algorithm 2 */
6: **if** $L == \emptyset$ **then**
7: **return** *failure*
8: **end if**
9: **return** L

genConf first attempts to satisfy information admissibility by checking on functionalities already existing in the configuration, whose relative execution time does not come after the functionality that we are trying to satisfy. Then, it attempts to satisfy information admissibility with the rest of the functionalities in **F**. Functionalities connected with information links are executed during the same time step.

If information admissibility is satisfied in a configuration, it stars checking for causal admissibility. *genConf* first checks if causal admissibility can satisfied by f_i. If it can be satisfied by f_i, then causal links are created. If there are still causal links to be satisfied, we choose one *targetcond* in *target* $\in F_c$, executing in timestep *targetime*, and we check if any functionality in the current configuration has a postcondition that can satisfy *targetcond*, as long as its execution time comes before the *targetime*.

On every case where a new connection is created, *genConf* calls itself recursively with the partial configuration c'.

6 Explanatory Example: Locating the Human

With this example, we intend to show two points. First, we want to show that it is possible to build admissible configurations just by interleaving the right information and causal links between functionalities.

This will be done without the need of HTNs, and without defining terminating functionalities. Second, we want to show that it is possible to define the goal

Algorithm 2 *genConf*. Partial configuration c, available functionalities **F**

Require: c,**F**
if inf. admiss. not satisfied **then**
  choose $target \in F_c$ with unsat. in. $targetIn$, executing in $targettime$
  for all $f \in F_c : targetIn \in O_f \wedge f_{executiontime} == targettime$ **do**
    $c' \leftarrow c$
    $append(ICX_{c'}, link(target, f, targetIn, targettime))$
    $genConf(c', \mathbf{F})$
  end for
  for all $f \in (\mathbf{F} \setminus F_c) : targetIn \in O_f$ **do**
    $c' \leftarrow c$
    $append(ICX_{c'}, link(target, f, targetIn, targettime))$
    $append(F_{c'}, f)$
    $genConf(c', \mathbf{F})$
  end for
else if causal admiss. not satisfied **then**
  choose $target \in F_c$ with unsat. prec. $targetcond$, executing in $targettime$
  for all $f \in F_c : targetcond \in Po_f \wedge f_{executiontime} \prec targettime$ **do**
    $c' \leftarrow c$
    $append(CCX_{c'}, link(target, f, targetcond, targettime, f_{executiontime}))$
    $genConf(c', \mathbf{F})$
  end for
  for all $f \in (\mathbf{F} \setminus F_c) : targetcond \in Po_f \wedge noConflict(Po_f, CCX_c)$ **do**
    $c' \leftarrow c$
    $append(CCX_{c'}, link(target, f, target, targettime, targettime + 1))$
    $append(F_{c'}, f)$
    $genConf(c', \mathbf{F})$
  end for
else
  $append(L, c)$
  return 0
end if

of a configuration not only as an action that needs to be performed, but also as information that we want to get from the world.

The state variables and functionalities available in the world are described in Tables 1 and 2. Our world can be described as the home of a person that needs some assistance in the daily life, with matters like localizing medications or health devices, tracking activities, or daily chores.

In this example, an information goal is accomplished with a configuration that contains direct information and causal links. The goal is to provide a position of the cell phone, which we bind to the position of the human. In the state variables, the position of the phone is unknown or outdated, so we need to acquire it again.

Table 1: State Variables.

Name	Value
bt (phone)	off
position (phone)	-
number (phone)	777

To get the position of the phone, Table 2 shows two functionalities that can help, which are $get-ph-GPS$ and $bt-tracker$. Functionality $get-ph-GPS$ obtains the position of the phone as GPS coordinates, if it has as an input the number of the phone. Functionality $bt-tracker$ obtains the position of the phone in terms of how close the phone is to a certain bluetooth source, if it has as an input the number of the phone, and if the

Table 2: Available Functionalities.

Id	Ins	Outs	Prec	Post
get-ph-nr	-	number (phone)	-	
get-ph-GPS	number (phone)	position (phone)	-	
bt-tracker	number (phone)	position (phone)	bt (phone) = on	
turn-phbt-on	number (phone)			bt (phone) = on

bluetooth of the phone is on. In the case of generating a configuration that uses $bt-tracker$, the phone bluetooth is off (Table 1). In order to satisfy causal admissibility, a causal link needs to be used. Such link can be done to functionality $turn-phbt-on$. This functionality turns on the bluetooth in a phone whose number is given as an input; $turn-phbt-on$ can be implemented by combining a program that runs in a local computer and sends an SMS with a code to the desired phone, and another program that when receiving the SMS with the code, turns on the bluetooth in the phone. In each case, the planner will call itself recursively until satisfying all admissibility criteria, reaching a limit size, or after recursively finding that there is no way to satisfy admissibility in any partial configuration, and then a failure is returned. For more details, see Section 5. In figure 3, the configurations obtained by applying our approach in this situation are summarized.

Execution monitoring is not part of this planner, but it if we had a system that monitored when a failure was found on applying a certain functionality, or a functionality is no longer available, then another program could update the description of functionalities, and our planner could be launched again with that feedback. However, a better approach for execution monitoring can be devised later on.

7 Conclusions and Future Work

In this paper, we have presented an algorithm for configuration planning in which functionalities can connect to each other directly by using either action or information links, without the help of a hierarchy definition, just by matching the information and causal needs of every functionality that is added into the configuration. Our planner can handle both causal goals and information goals. The combination of this features allow the execution of an action in order to fulfil an information requirement, and the capture of an information piece in order to execute an action.

Our current approach has a number of limitations that we would like to overcome. However, we would like to keep the idea of direct information and causal links in a configuration and we would like keep the idea of defining goals as interactions with the world or as information pieces to obtain. But we would like a more expressive way of defining goals, and of defining how information and causal requirements relate with the execution time of the functionality; an example of the latter is being able to discern if a functionality changes a state variable or gives an information output either during its execution time, after execution, or from the start of its execution. Also, the planning in the causal dimension in our proposed algorithm, is based on causal links planning without any guidance of heuristics. In order for our planner to scale up to more causally challenging problems, this needs to be addressed.

For the application of our work in the **Giraff+** Project, we want to handle limited resources, multiple goals with preferences, and interactions with activity recognition. More particularly, we are interested on describing preferences for reliability, quality, performance, use of resources and handling of undesired consequences. We are interested on studying interactions between multiple requests, requests that can not be fulfilled because of a conflict or resource scarcity, and on handling disjunctive requests. For challenges associated to real application domains, we are interested on seamless ways of adding and removing functionalities in the system, on handling failures and degradation of performance in configurations, and on execution monitoring.

In order to address many of the previous matters, for the near future we are defining a language for expressing goal preferences, as well as interactions of time with causal and information requirements in functionality definitions. We are also defining heuristics and will use semiring soft constraints for guiding the construction of configurations in a smarter way.

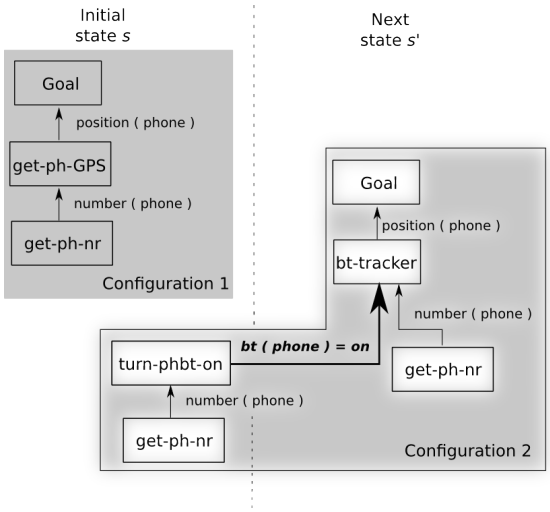

Figure 3: Configurations on Example 'Locating the Human'

Acknowledgements Giraff+ is funded by the European Community's Framework Programme Seven (FP7) under contract #28817

References

[1] Giraff+ website. http://www.giraffplus.eu/.

[2] M. Gritti, M. Broxvall, and A. Saffiotti. Reactive self-configuration of an ecology of robots. In *Proceedings of the ICRA-07 Workshop on Network Robot Systems.*, 2007.

[3] X. Su. J. Rao. A survey of automated web service composition methods. *Semantic Web Services and Web Process Composition. Lecture Notes in Computer Science, Springer*, 3387:43–54, 2005.

[4] LSilvaWebsite. http://aass.oru.se/ ˜lcsz/.

[5] R. Lundh. *Robots that Help Each Other: Self-Configuration of Distributed Robot Systems*. PhD thesis, Orebro University, 2009.

[6] R. Lundh, L. Karlsson, and A. Saffiotti. Autonomous functional configuration of a network robot system. *Robotics and Autonomous Systems, Elsevier*, 56(10):819–830, 2008.

[7] F. Pecora, M. Cirillo, F. Dell'Osa, J. Ullberg, and A. Saffiotti. A constraint-based approach for proactive, context-aware human support. *Journal of Ambient Intelligence and Smart Environments*, 2012. (To appear).

[8] J. Peer. A PDDL based tool for automatic web service composition. *Principles and Practice of Semantic Web Reasoning, Lecture Notes in Computer Science, Springer*, 3208:149–164, 2004.

[9] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2010.

[10] 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)*, pages 275–280, 2005.

[11] M. Tang. A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition. In *Proceeding of the 2010 IEEE World Congress on Computational Intelligence*, 2010.

[12] J. Ullberg, A. Loutfi, and F. Pecora. Towards continuous activity monitoring with temporal constraints. In *Proceedings of the Workshop on Planning and Plan Execution for Real-World Systems at ICAPS09*, 2009.