| | |
|---|---|
| SEVENTH FRAMEWORK PROGRAMME | **Project Acronym: Giraff+**<br>**Project Title: Combining social interaction and long term monitoring for promoting independent living**<br>**Grant agreement no.: 288173**<br>**Starting date: 1st January 2012**<br>**Ending date: 31st December 2014** |


# D3.3 Fully functional reasoning infrastructure

| | |
|---|---|
| **WP related to the Deliverable:** | 3 |
| **Nature:** | P |
| **Dissemination Level :** | PU |
| **Version:** | V0.4 |
| **Author(s):** | Lars Karlsson, Amy Loutfi |
| **Project Participant(s) Contributing:** | ORU |
| **Contractual Date of Delivery:** | 20140630 |
| **Actual Date of Delivery:** | 20140721 |

## Document History

| Version | Date | Type of editing | Editorial |
|---|---|---|---|
| 0.1 | 13/06/14 | Table of Contents | ORU |
| 0.2 | 29/06/14 | First complete draft | ORU |
| 0.3 | 16/07/14 | Second expanded draft for internal review | ORU |
| 0.4 | 21/07/14 | Final version | ORU |

## Deliverable Summary

This deliverable contains material that supports the delivery of the fully functional reasoning infrastructure. The latest version of the context recognition module is now operational in the test sites, and an alarm function has been added to it. There has also been an initial small-scale user evaluation of the context recognition, and an engineering tool for validating queries has been developed. There is a second version of the configuration planner which supports reasoning about preferences in order to find configurations that are more adapted to the needs of users.
The context recognition and the configuration planner are now integrated through the context recognition preprocessing module: the latter is configured by means of an XML file which describes what sensors to use and how to preprocess that sensor data before the context inference is applied. The configuration planner, given a description of what activities needs monitoring, then produces a new XML files in order to enforce a reconfiguration. This also makes the deployment of a configuration straightforward, as everything is handled inside the preprocessing module.
Finally, we have developed a collection of statistics for long-term trend analysis. These statistics will be integrated with the DVPIS (developed in WP4).

S

# Contents

# 1 Introduction

## 1.1 Scope of the document

The following document constitutes deliverable 3.3 in the GiraffPlus project. It presents the progress done since the previous deliverable, 3.2 from month 18.

## 1.2 Deliverable structure

The deliverable is structured according to the five tasks comprising Work package 3:

1. *Long-term context recognition* where during the past year the focus has been on preprocessing of sensor data, on providing an engineering interface for validating queries, and on-site testing with a secondary user.
2. *Configuration planning for efficient activity recognition* where the focus has been on developing a new planner which can take into account multiple categories of preferences.
3. *Integration of context inference and configuration planning* where there is a completely new integration utilizing the preprocessing module of the context recognition.
4. *Long-term trend analysis* which started from month 18, and for which some statistics have been developed which are to be integrated into the DVPIS developed in WP4.
5. *Deployment* where the deployment of configurations generated by the planner has been made easier by taking advantage of the context recognition preprocessing module.

## 1.3 Deviations with respect to the plan

The initial plan intended for the long term trend analysis to be integrated with the context recognition. After first implementation and understanding of how the long term trend analysis was to be used, it is instead being integrated with the DVPIS. This provides an easier possibility to visualise trends and conceptually makes sense with respect to the function of the DVPIS as a visualisation tool.

Besides that, the work in WP3 progresses according to plan.

# 2 Long-term context recognition

The context recognition and its main components have been deployed at the first round of testsites, and since M18, the main efforts in this task has been focussing on improving the operation of the context recognition module and the eventual integration with other components in the system, like the configuration planner (as will be described later). In short, the main improvements during this reporting period are:

- Developing better support for generating multiple time lines from sensor data - this task involves using more expressive rules to trigger events and alarms which can now depend on multiple sensor modalities simultaneously. This is described in section 2.1
- Adding an alarm service that is raised based on the output from the context recognition. This is described in section 2.4.
- Developing an engineering tool for validating context recognition queries.
- Applying and testing the context recognition in the GiraffPlus system and in particular on the test sites. In section 2.6, we describe an initial evaluation with a secondary user. This is an important step considering the user focus of the project. As this evaluation concerns a very specific part of the GiraffPlus system, it is done here in WP3 and not WP6.

Recall that in GiraffPlus context recognition is the key enabler to automated behavior monitoring over time. As such, the monitoring solutions developped in this project must possess two key qualities: (**requirement 1**) the ability to selectively focus on different aspects of daily life depending on circumstances that are assessed by a physician or family member; and (**requirement 2**) the ability to trace these aspects over medium to long periods of time. Interesting health-affecting behaviors can be: decrease of physical activity, irregularity in sleep, changes in cooking and eating habits and so on.

WP3 presents a context recognition system that addresses the two requirements above. The system infers and records the activities and status of elderly over extended periods of time. The specific way in which behaviors are recognized are specified through *temporal models*, which can be defined, added or removed dynamically to a list of behaviors of interest. Caregivers, medical experts and family members (henceforth, *secondary users*) can search and inspect the recorded information through a versatile user interface which supports real time viewing of what is happening in the elderly person's home. The interface also aggregates and provides tools to analyze data extending over long periods of time. The models used for behavior tracking are specified in the form of qualitative relations among sensor readings. During this reporting period, we have assessed through preliminary evaluations with secondary users that these qualitative relations are intuitive for secondary users. The specification syntax used currently is based on XML. A primary goal for exploitation after the project will be to asses whether a more user-friendly input method (e.g., graphical) should be developed to enable secondary users to input these relations directly without training.

In addition to providing behavioral traces for secondary users, the context recognition infrastructure also synthesizes appropriate action plans to aid the primary user when certain conditions hold on the recognized behaviors. Such enactments manifest themselves as proactive alerts, e.g. if the user is recognized as having altered sleep patterns over several days, appropriate physiological and activity-related information is presented to the caregivers.

It is important to restate here that the context recognition engine used in the GiraffPlus system is mostly related to temporal constraint-based approaches such as SAM [6] and constraint-based chronicle recognition [3]. These approaches employ temporal reasoning techniques to perform on-line recognition

of temporal patterns of sensory events. An approach based on evidence theory augmented with temporal features presented by [4] underscores the advantage of explicitly accounting for activity durations. GiraffPlus introduces a key novelty in temporal constraint-based context recognition, namely the ability to take temporal uncertainty in the sensor readings into account. This capability is an important enabler of configurable (**requirement 1**) and continuous (**requirement 2**) recognition, as this allows us to interpret the output in time of sensors in ways that fit high-level, user-defined models of behavior, and possesses the necessary good performance to be used on-line.

In summary, GiraffPlus extends the state-of-the-art in context recognition in terms of (1) models of human behaviors that are instantiated on-line, (2) generalization of activity recognition to context recognition by taking multiple sources of physiological and environmental data into account, and (3) applicability to real world scenarios.

In the GiraffPlus system, the context recognition is implemented as a REST service that runs as a servlet on a central Tomcat server. Queries to this service are done with a lightweight API which is embedded in several services that run on the client computer or on the central server. All computations are done on the central server when querying an activity. This architecture has the advantage of;

- Reducing bandwidth (since it does not need to transfer raw samples across the network).
- Allowing for a more strict access control to sensor data.
- Enabling system updates without requiring changes to client software.

The context recognition service has an inference procedure that is divided into three distinct steps: preprocessing, inference and extraction. The responsibilities of these are described in the following sections. We also describe the alarm service, the query validation tool, and an initial test-site evaluation of the context recognition.

## 2.1 Preprocessing module

On the server the client sends a query to the context recognition engine by providing it with an XML-document describing how sensor data and activities correlate. The preprocessing module's responsibility is to fetch samples from the database and use these samples to build a higher level representation of the events that takes place in the home. This is done by using an appropriate preprocessor for the data. For instance, a time line that declares if a person is at a location or not, based upon PIR-motion sensors, can be constructed either by looking at individual sensors or by using sensors at other locations as well as terminal conditions. In the former case a temporal threshold parameter needs to be provided to determine the temporal extent to which a person is considered to be in a room, for this to work a continuous sequence of repeated motion readings needs to be generated by the user, and the query is parameterized with with a maximum allowed temporal discontinuity between these. In the latter case the person is considered to be at a location until he is sensed somewhere else.

Since month 18, the major contribution in developing the context recognition system has been towards solving the problem of how to go from sensor readings to sets of temporal intervals, which are then used by the inference algorithm described in the next section. For this purpose, we have developed a library of procedures for generating the inputs of the inference algorithm. Among these, there are primitives to pre-compute sets of intervals not just from one sensor trace but many. An example of where this is useful is to determine room occupancy: the presence of the user in room X is determined by the PIR sensor in room X being "on", AND the PIR sensors in other rooms being "off". This allows to make a more precise determination of when the user should be considered as being in a room, and makes this information more robust with relation to the (imprecise) placement of the sensors in the primary user's home. In addition, it is possible to provide different semantics to different sensors, e.g., once the user is in room X, a triggering of other indicators of presence in another room is necessary to discontinue the presence of the user in room X (intuitively, because a user may go to a room and then stop moving, which should not mean that the user is no longer in the room).

The library of pre-processors specifically caters to the diverse conditions and environments in which the GiraffPlus system is deployed (e.g., making use of all multiple PIR sensors for determining presence of the user in a room will make this sensor reading robust to sensors that are moved as a result of dusting, or that have broken over time.)

We are presently working on going from the hard-coded procedures and ad-hoc semantics for obtaining accurate interpretations of sensor readings, towards a more systematic approach in the form of a well-founded "timeline logic". This will increase the versatility and expressiveness of the rule specification language, e.g., by allowing the negation, disjunction and conjunction of sensor traces.

As will be shown in section 4, the configuration planner realizes a configuration by updating the XML-document for the preprocessing module of the context recognition.

A complete example of an XML file with preprocessing (as well as inference and extraction) can be found in appendix A.

## 2.2 Inference module

The symbolic models underlying the inference are grounded on a *constraint-based representation*. The key advantage of doing so lies in the widely recognized capability of this paradigm to support search and incremental constraint solving capabilities, and the relative efficiency of the resulting applications. The user-supplied rules used by the inference module define how sensor readings correlate to context that can be inferred. These correlations are expressed as temporal constraints in Allen's Interval Algebra [1] with metric bounds, however, the overall architecture supports the more expressive INDU algebra [7] which adds constraints on the relative duration of intervals. Activities are inferred by performing temporal constraint propagation on the domains ationof intervals generated by the preprocessing module and the output is a domain of intervals that are admissible with respect to the rules. The propagation and inference algorithm is described in detail in deliverable 3.1. and in [9].

## 2.3 Extraction Module

The extraction module's responsibility is to generate time lines that can be used by other software components (e.g., the visualization software or the alarm system). As the inference and preprocessing module generates large amounts of hypotheses about the activities that have taken place there is the need to provide a system to easily analyze this data. Presently, this module only supports one type of extraction method, which extracts the maximum duration interval for an activity.

## 2.4 Alarm Checker

This system regularly queries the context recognition module for user-defined alarm conditions. If an alarm condition is detected the system will send a Pushover notification to alert relatives and caregivers.

The Pushover Client runs on Android and iOS devices and presents short messages to the user. There are different levels of urgency to these messages which controls if the receiver is alerted with a sound during night or not for instance.

A screen shot from the Pushover client is shown in Figure 1.

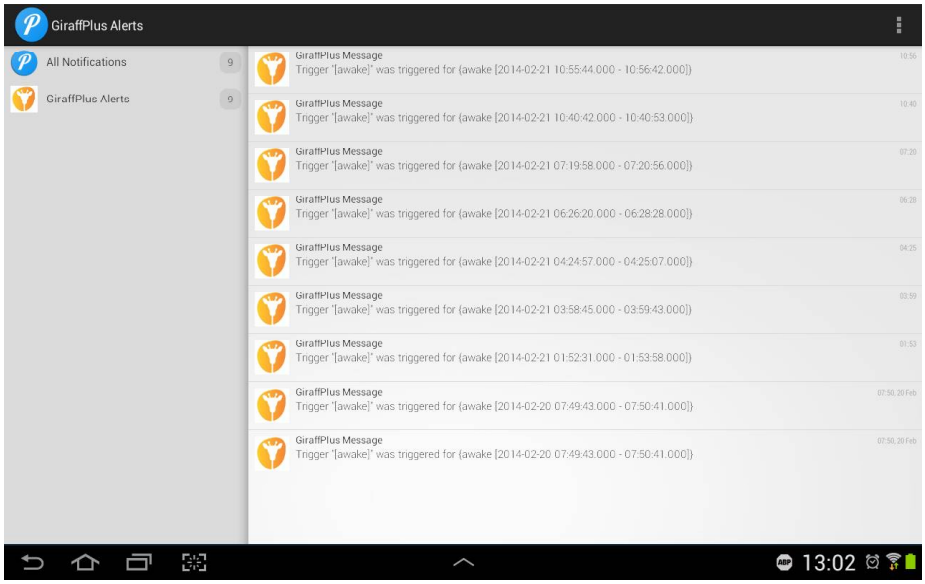


Figure 1: Screenshot of the pushover client when an alarm is set to trigger for an awake activity

This alarm feature via pushovers is an additional feature beyond the original scope of the project. It was decided to use this type of service in order to enhance the currently possibilities to provide alarms to users. Deliverable 4.2 also details other means of generating alarms via the visualization tool. The alarm checker is mentioned here as it relates directly to the context which is inferred by the context recognition.

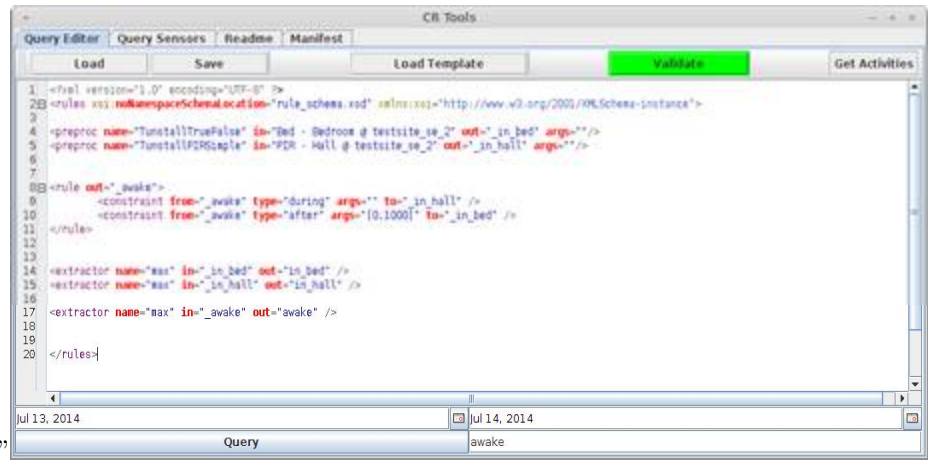


Figure 2: Screenshot of the Query Tool

## 2.5 Query Tool for Context Recognition

In order to deploy the Context Recognition as described in D3.1, a number of tools were created within the scope of this workpackage. It became apparent after the first round of test site installations, that tools for monitoring and verifying rules were required for the engineers. Such tools should be differentiated from the DVPIS which is really aimed at visualization for end users. Rather, the tools developed in WP3 are there for the developers of the context recognition and are designed to be used together with other project members. Therefore, we will summarize the main features of the tool and describe how it operates.

One of the novelties of the context recognition algorithms developed in GiraffPlus is the possibility to re-query the data online using different activities of interest which emerge at a later stage in time. This is highly useful as knowledge that is acquired during the deployment of a system can be used to re-assess data. Also, this is useful from the point of view of creating an adaptive system w.r.t. end users whose physical or cognitive health may degrade or improve over time. A challenge however, with such a system is that it is important to be able to check and validate queries. At this point in the project, the language of the queries are still rather technical, in the sense that the one performing the query must have some basic knowledge of how to form requests using the proper Allen's interval logic. Clearly a direction for the future is to develop a proper querying language that is intuitive for all users. For now, the following querying tool is intended for internal use to verify the results of the Context Recognition algorithm. The main panel of the query tool/rule editor looks as shown in Figure 2.

**Load** : This allows the GP technician to load rules. Rules are contained in .cr files and once loaded appear in the main window. The rules, as explained before, are written in XML and contain a preprocessing part and a rule expressed with constraints that are mapped to activities. In the figure above, the rule awake has two contraints mapped to two temporal concepts. The first, states that there should be movement in the hall after it has been detected that the person has arisen from the bed. In this tem-

plate only one activity, awake, can be detected. However, it is possible to load templates where many activities can be detected. The list of relevant activities of course depend on the end users profile.

**Validate** : Checks that rules are syntactically correct. In other words, it verifies that the rule file follows the predefined language. For example it will put constraints on the language by requiring that e.g. variable and references to the sensor names are correctly defined. This means that it not only has to be present but also provided by a preprocessor `<preproc>`. This function also checks the correct embedding of the XML tags (well formed and schema).

**Query Sensors** : A function that retrieves the names of all sensors given a specific test site. This facilitates defining the rules. Get Activities: Scans the document and extracts the names of the possible activities that are defined in the main panel and lists them in text bar at the bottom of the screen. The activity list can be edited after if the user does not want to query for all listed activities.

**Query** : Sends the query to the Context Recognition Server. The results for a query are then presented in timelines (similar to how they are presented in DVPIS).

A download of the query tool can be found here:
$https://dl.dropboxusercontent.com/u/636027/querytool_0.0.7.jar$

## 2.6 Context-recognition initial evaluation

During the first half of 2014, we have made an initial effort to evaluate whether the context recognition service can infer time lines that gives a realistic and useful picture of the old persons' activities. Since it is difficult to collect ground truth of performed activities (due to the fact that the elderly can't realistically be asked to annotate everything they are doing) an evaluation was done together with a local caregiver with insights into a test subject's daily life and medical history. The goal was to assess how well the system could infer medically meaningful information about the users daily life. We intend to extend this evaluation to involve other test sites and other secondary users.

The apartment in this case study is inhabited by an 82 year old man (born 1931) which has been living alone since his wife passed away two years ago. At around the same time the man had a stroke and spends most of his time inside, the exceptions are when he goes outside to do shopping or to visit any of his three sons with his mobility scooter. The man receives help from home care four times a day that ensures that he is feeling well and that he takes his medication. The man's sons live nearby and visits him often, and his grandchildren use the telepresence robot to visit him remotely.

The man's apartment is depicted in Fig. 3. Before deploying the system in the home the inhabitant was interviewed. The answers given during the interview was used to determine a good sensor placement that would allow the system to capture as meaningful traces of his daily activities. This resulted in the fact that the laundry room and the study were not instrumented at all since the inhabitant almost never used these, and the living room was sparsely

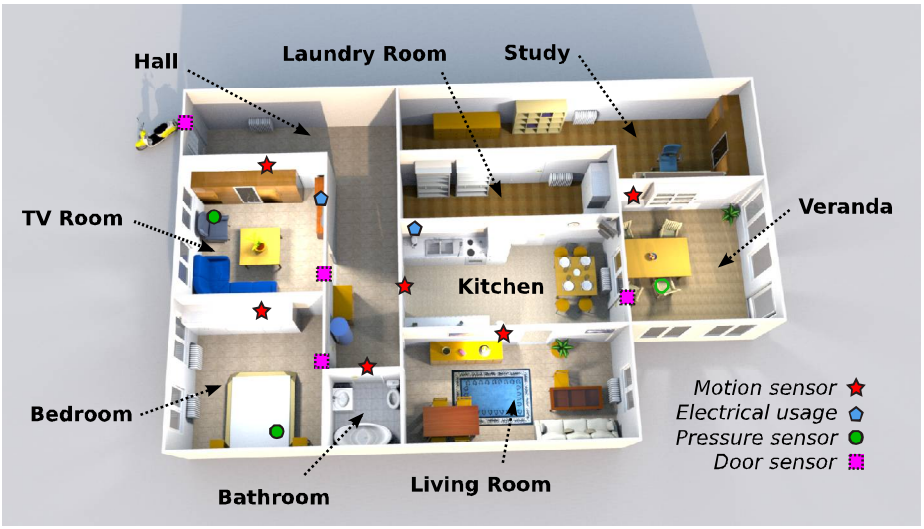


Figure 3: The layout of the second test site in Sweden. This is not an exact depiction but captures the general layout of the large home.

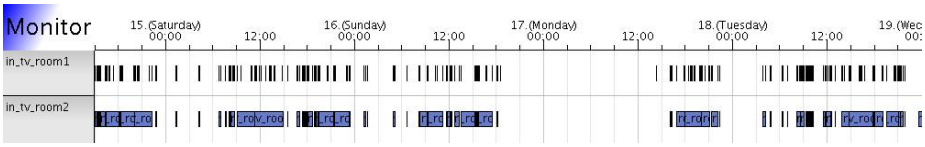


Figure 4: A pair of time lines showing when the elderly man visits the TV room, constructed using different methods of preprocessing the sensor data.

instrumented since it was only used when the man had visits. Conversely, the TV-room, the kitchen, the bathroom and the bedroom were considered important and therefore equipped with more sensors.

The session with the caregiver resulted in several queries to the context recognition system using a horizon of two weeks[1]. In the beginning of the session the caregiver claimed that the man had stated that he spends much of his time in front of the TV. The caregiver wanted to know how often and when the person was watching the TV since this behavior can influence his health. Consequently, a query was made to see how much time the user spent in front of the TV using the motion sensor in the TV room[2]. The time lines resulting from this query are shown in Fig. 4.

The topmost time line, `in_tv_room1`, in Fig. 4 shows the result of the first query. Given the fragmented nature of the time line (containing many short intervals) it appeared as if the person was mostly sitting still in the TV room, or at least not moving enough to trigger the motion sensor frequently enough to generate continuous intervals on the time line. In order to address this problem,

[1] A more limited timespan was chosen for the graphics used in this report so that details are visible.

[2] The motion sensor was used instead of the electrical usage sensor connected to the TV since the former appeared to be in an always on state. We suspect this happens because the TV consumes enough electricity in standby mode to be considered on.

```
<rules home="testsite_se_2">

<preproc name="TunstallPIRSimple" in="PIR - TV Room"
   out="_in_tv_room1" args=""/>
<preproc name="TunstallPIRSimple"
   in="PIR - TV Room, PIR - Bedroom, PIR - Kitchen"
 out="_in_tv_room2" args=""/>

<extractor name="max" in="_in_tv_room1" out="in_tv_room1" />
<extractor name="max" in="_in_tv_room2" out="in_tv_room2" />

</rules>
```

Figure 5: A rule that infers when the person has been in the TV room using two different methods.

another query was made using data from other motion sensors in the apartment as well, the output of this query is shown bottommost in Fig. 4 as `in_tv_room2`. Here, the data from the additional sensors were used as terminal conditions for ending the activity (the motion sensor placed in the hall adjacent to the TV-room was particularly important). The time line for `in_tv_room2` is clearly more continuous than `in_tv_room1` but still contains some discontinuity. This is probably due to a bad placement of the motion sensor in the hall, allowing the user to be detected even though he is in the TV-room. At some occasions this can also be due to the fact that he had had visitors, e.g. home care or relatives, as they move around the apartment they constantly end the `in_tv_room1` activity.

One responsibility of the context recognition module within GiraffPlus is to provide time lines containing performed activities to a statistics extraction module, the result of the second query forms a much better basis for assessing time spent in front of the TV during the day and can be used over longer horizons to detect changes in behavior and anomalies. The rules created to detect when the person is in the TV-room is shown in Figure 5.

Even though these queries did not produce optimal visual results, the caregiver had gotten a better understanding of the persons habits, and it can clearly be seen that the person spends many hours a day in front of the TV. Also, the caregiver noted that the man's TV-watching habits were not isolated to daytime. After having inspected the man's TV-watching habits, the caregiver was interested in the evening and night time activities of the man since he could be seen to watch TV late at night at some occasions e.g. on Sunday the 16th. In addition, discussions with the person had revealed that he sometimes went up during the night to read the newspaper in the kitchen.

As the evaluation session continued the caregiver wanted to see when the person went up at night to look at the TV or to read the newspaper so rules were constructed to filter out these events. In addition to processing the sensory data, a rule that filters out events where the person had left the bed and went to either of these locations were constructed using the language of Allen's Interval Algebra. Activity intervals `awake_in_kitchen` and `awake_in_tv_room` were inferred on a time line so that each filtered interval occurred `AFTER` `in_bed` and `DURING` presence at the respective locations; `in_kitchen` and `in_tv_room`.

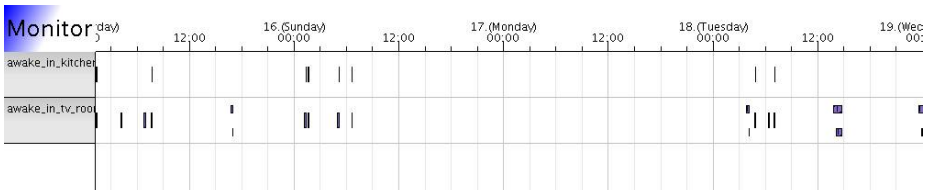


Figure 6: A pair of time lines showing when the old person visits the kitchen and the TV-room after having left his bed.

```
<rules home="testsite_se_2">

<preproc name="TunstallTrueFalse" in="Bed - Bedroom" out="_in_bed"/>
<preproc name="TunstallPIRSimple" in="PIR - Kitchen" out="_in_kitchen"/>

<rule out="_awake_in_kitchen">
<constraint from="_awake_in_kitchen" type="during"
   to="_in_kitchen"/>
<constraint from="_awake_in_kitchen" type="after"
   args="[0,1000]" to="_in_bed"/>
</rule>

<extractor name="max" in="_awake_in_kitchen" out="awake_in_kitchen"/>
...
```

Figure 7: A fraction of a rule that is used to determine which rooms the elderly visits when he leaves his bed.

The output of this query is shown in Fig. 6.

It can be seen that the user typically visits both the TV room and the kitchen when he leaves his bed. Also, this behavior seems to be a part of a habit since it occurs so often. A fraction of the rule document created to detect when the person leaves his bed to visit the TV-room and the kitchen is shown in Figure 7. There are two preprocessors that take sensor data (via the database) from two specific sensors (a bed pressure sensor and a PIR sensor in the kitchen) convert them into Boolean values, and store the results in two state variables `_in_bed` and `_in_kitchen`. Then there is a rule for inferring the activity `_awake_in_kitchen` from these two state variables. This activity has multiple interpretations (i.e. sets of intervals). Finally, there is an extraction method which selects the interpretations with the maximal intervals and stores the result as the activity `awake_in_kitchen` (without initial underscore). It is the latter that is the end result of the context recognition process.

To obtain a verification of the inferences produced by the system, the results were discussed by the elderly man. He confirmed the inferences with his own recollection of his activities. During this discussion the man expressed discomfort about the system knowing how often he had been awake during the night. Despite being well informed of the system's capabilities, he expressed that he was less comfortable with an aggregation of long term data about his habits than with alternative technologies such as observing him visually from time to time through a video camera.

# 3 Configuration planning for efficient activity recognition

Recall that the purpose of the configuration planner in the GiraffPlus system is to provide automatic configuration of sensor data retrieval and processing for different activities in different apartments with different available sensors (or just differently labeled sensors) at different times. The goal is to provide the data needed for inferring the requested activity. One advantage is that the inference rules of the context recognition can be decoupled form the specific set of available sensors and the specific labels those sensors have been given in a specific apartment.

Between month 18 and month 30, the main focus on the configuration planning task has been towards developing a second version of the planner which can take preferences between configurations into account. These preferences should be used to select between different alternative configurations for the same task according to different criteria. These criteria may take into account e.g. that different sensors can provide data of different quality for the same observable variable or may consume different amounts of energy, or the same sensor may perform differently under different environmental conditions, that processing functionalities may perform differently depending on the source of their input, or that primary or secondary users prefer certain sensors/processing under certain conditions. For instance, whether a person is in the living room can be determined with higher reliability by combing data from both motion detection (PIR) and a pressure sensor in the sofa, than by just using data from one of these sources. Thus, configurations which rely on both sensors might be preferred to configurations than rely on only one of them. However, the configuration planner should also be able to propose the less preferred configurations in situations when only one type of sensor is available.

The extension to preferences presents two major challenges: (1) how to represent, compute and aggregate preferences, and (2) how to optimize according to multiple preference categories. In the following, we first briefly describe the first version of the planner, and then the novel contributions of the second version.

## 3.1 First version

The first version of the planner was inspired by causal-links planning. It took as input (1) a set of available functionalities (sensors, actuators, programs) with their information inputs and outputs and causal preconditions and effects (2) the present causal state and (3) a set of information and/or causal goals. It worked by improving a partial configuration by repeatedly selecting an unsatisfied goal/precondition/input and then selecting a functionality that hand an effect/output could satisfy it. A causal/information link was also created between the effect/output and the goal/precondition/input. Causal links implied sequential execution and information links implied concurrent execution of the connected functionalities. This went on until an admissible configuration (i.e. one with no unsatisfied preconditions/inputs) that also satisfied the goals was found.

The algorithm started with an empty configuration and the initial state and goals, and then expanded from there, maintaining a search front of partial

configurations and a set of found solutions. We investigated empirically different heuristics inspired by heuristics used for constraint satisfaction. These heuristics were used to (1) select which unsatisfied goal/precondition/input to first work on in a partial configuration and (2) select which partial configuration in the search front to improve (see Deliverable 3.2) An article about the planner and the empirical investigation is currently under a second round of reviewing for *Journal of Ambient Intelligence and Smart Environments*.

## 3.2 Second version

Since autumn 2013, the work on task 3.2 has focused on developing a configuration planner which can take into account preferences. A configuration planning problem often has more than one solution (configuration). In the first version of the planner, the only criteria for choosing between alternative solutions were the number of functionalities and links. This second version is intended to provide a framework for specifying preferences between configurations. This will make it possible to obtain configurations that are more tailored to the needs of primary and secondary users, as well as take into account the strengths and weaknesses of the available sensors, actuators and programs. For an overview of the related topic of task planning with preferences, see [2]. In this section, we present the underlying representation of preferences, the multi-objective criteria used by the planner, the new algorithm, and the current status of the experimental evaluation.

### 3.2.1 Preferences

There can be a number of categories of preferences, e.g. reliability or cost. Each category of preferences takes a value from a c-semiring with values, for instance the range 0..1. A semiring is an algebraic structure containing a set $A$, and two binary operations corresponding to addition and multiplication.

$$X = \langle A, +, \times \rangle$$

In X, the addition operation $+$ is commutative with identity element 0 and closed, the multiplication operation $\times$ is associative, closed, distributes over $+$, and has 1 as the unit element and 0 as the absorbing element. When a semiring has idempotency (for all $a \in A: \ a + a = a$), a commutative $\times$ operation, and 1 as the absorbing element in $+$, then such semirring is a c-semiring in which $\{0, 1\} \subseteq A$.

$$S = \langle A, +, \times, 0, 1 \rangle$$

The preferences introduced into the second version of the planner are used as follows:

- Preference values can be specified for outputs (sources) and causal effects. The planner selects solutions with more preferred outputs and causal effects.
- Lower limits for preference values can be specified for satisfying preconditions, inputs (sinks) and goals. The planner eliminates solutions where a precondition/input/goal is satisfied with an effect/output which has a preference value below the given limit.

Values/limits can be given for each category separately. In addition, the values can be conditional on the current state.

The following is a simple example of two functionalities, one of which is a sofa occupancy sensor and one which determines the presence of a person in the livng room using the information output (`source`) of the first one as input (`req`):

```
sofa_occupancy_sensor {
  source { sofa.occupancy.bool { reliability 0.8 }}
}

check_in_livingroom_1 {
  req { sofa.occupancy.bool {} }
  source { livingroom.presence.bool { reliability 0.2 } }
}
```

Note that although the sofa occupancy sensor is typically preferred (0.8 for reliability in the first functionality) for determining whether somebody sits in the sofa, it is typically not preferred for determining presence in a room (only 0.2 for reliability in the second functionality).

For a given configuration $x$, an aggregated value (typically with the $\times$ operation) for each preference category can be computed. We denote his value $f_i(x)$ for the $i$th preference category. Together, these values form a vector $(f_1(x), ..., f_n(x))$.

### 3.2.2 Multiple-objective optimality criteria

As there are multiple category values $(f_1(x), ..., f_n(x))$ for each configuration $x$ and each of those values should be maximized, configuration planning with preferences becomes a multi-objective optimization problem. In order to determine when a configuration is preferred relative to another one, one needs to define a dominance relation in which weak preference is denoted $\succeq$; strict preference $\succ$ is defined as $x \succ y$ iff $x \succeq y$ and not $x \succeq y$; and finally indifference $\cong$ is defined as $x \cong y$ iff $x \succeq y$ and $x \cong y$. In the current implementation, two different dominance relations over configurations $c \in \mathcal{C}$ are used:

- The Pareto dominance relation $\succeq_{\mathcal{P}}$: $x \succeq_{\mathcal{P}} y$ iff $f_i(x) \geq f_i(y)$ for each preference category $i$.
- The Lorenz dominance relation $\succeq_{\mathcal{L}}$: let $(f_{i_1}, ...., f_{i_n})$ be the category functions sorted in increasing order. We define the generalized Lorenz vector as $\mathcal{L}(x) = (l_1, ..., l_n)$ where $l_j = \sum_{k=1}^{j} f_{i_k}(x)$, $1 \leq j \leq n$. $x \succeq_{\mathcal{L}} y$ iff $\mathcal{L}(x) \succeq_{\mathcal{P}} \mathcal{L}(y)$ [5].

With these criteria, one can identify solutions that are optimal relative to multiple objectives (i.e. preference categories), which means they are not dominated by any other solution. For $\succeq_{\mathcal{P}}$, these are the solutions that cannot be improved relative to some preference category without worsening some other preference category. The Lorenz dominance relation is similar, but also tends to favor more balanced solutions.

### 3.2.3 Algorithm

Much of the algorithm is the same as in version 1, although it has been completely reimplemented in order to make it more efficient. The major additions are that:

1. Preference values/limits can be added to the elements of a functionality and a goal.
2. Aggregate preference values for the different categories are computed for configurations (both partial and complete).
3. The set of solutions only contains non-dominated (Pareto or Lorenz) solutions.
4. The search front is pruned of non-dominating partial solutions.

We present pseudocode for the algorithm in appendix B.

We are now performing a thorough experimental evaluation of our algorithm configuration planning with preferences, by automatically generating artificial test domains with specific properties such as the average number of requirements and sources for functionalities. This allows us to in particular test the scalability of the algorithm under different kinds of domains, including domains that have different properties than GiraffPlus. We also intend to perform online testing of the configuration planner in the GiraffPlus system.

# 4 Integration of context inference and configuration planning

In the first version of the integration, the context recognition service and the configuration planning service were two processes that exchanged information through a socket. The configuration planner was responsible for setting up subscriptions and starting different programs for data processing. The first integrated system was described in an article presented at the *Ambient Intelligence* conference [8].

In the new integration, we take advantage of the preprocessing module, which can request data from the storage, preprocess this data in various ways and then make it available to the inference module. As mentioned before, the configuration of the preprocessing module is specified in an XML file. An example is provided in appendix A. Hence, the new version of the configuration planner has been given the capability to output configurations as XML-files. In that way, the integration has been significantly simplified, and it is easy to run the system both with or without configuration planning (the latter can be useful for instance for testing).

In addition, the configuration planner can now take activities, and not just observable state variables, as goals. It then uses a table that has been extracted from the context inference rules which maps activities to sets of state variables, as well as a table that maps state variables to specific sensors available in the apartment.

A configuration is encoded as an XML file as follows:

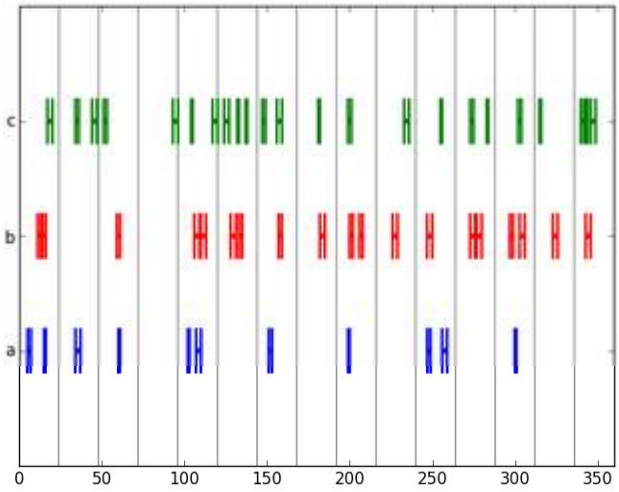


Figure 8: Time line line divided into epochs, with activities.

- Sensors are connected to state variables by means of preprocessors. These preprocessors may may either just copy the data or perform some kind of initial processing.
- Processing functionalities are preprocessors that read from state variables and produce further state variables.
- Channels between functionalities are represented as intermediary state variables.

# 5 Long-Term Trend Analysis

This task started after month 18. In this section, we present the work done so far for developing suitable statistics for activity time-lines. We believe that such statistics can be useful for secondary users that are looking for long-term changes in activity pattern, such as a person going to bed later, or spending more time in front of the TV.

As an example, figure 8 shows a randomly generated time line with three different activities (a, b c). The time line is divided into 15 epochs of 24 time units each, delimited by thin vertical lines. An epoch may e.g. be a day, and the time unit would then be hours in this figure.

Figure 9 shows a statistic - sum of durations in each epoch - for each one of the activities. This is computed by splitting the time line into epochs so that each epoch contains a set of intervals. Then the duration of the intervals is summed up for each epoch. Figure 9 also includes a linear trend line for each activity, in order to indicate the long-term change. This may otherwise be difficult to see if the statistic varies a lot between epochs.

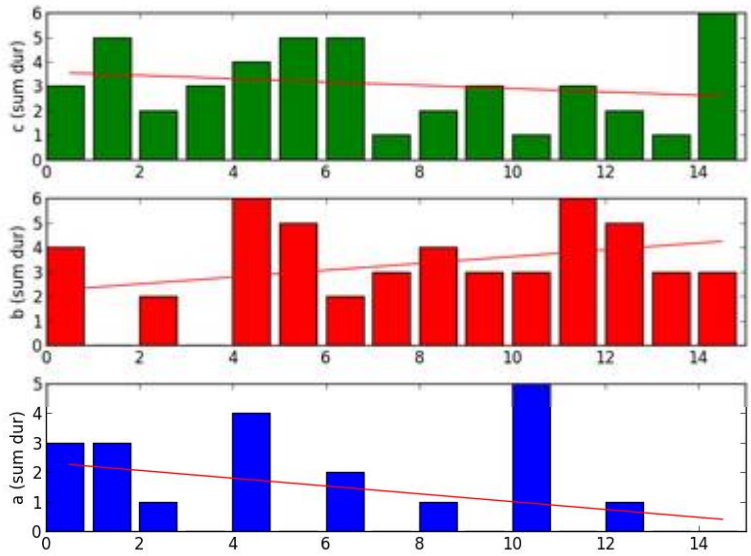


Figure 9: Statistics (sum of durations) for activities from figure 1. The x axis is marked with the epoch. Linear trends are also included.

Figure 10 shows three different statistics for a single activity. Again, the statistics are computed epoch-wise, and a linear trend line is included.

Figure 11 shows a scatter plot of two epoch statistics. These kinds of plots are used to test for correlations between different activities. In this case, the correlation is weak, as indicated by the red regression line and the large distance between the line and many of the points. An $R^2$ value could easily be computed in order to put a number on the degree of correlation.

The above is based on applying standard statistical techniques to timelines, but could still be very useful for a secondary user who wants to detect long-term changes in activities. Thus, the aim is to provide an interface as part of the DVPIS (see deliverable 4.2) for generating graphs like those in figures 9 to 11 for selected activities, statistics, epochs and time periods according to the needs of the secondary users. This can be seen as a complement to the long-time summary reporting in the current version of the DVPIS.

The above graphs were computed and generated using Python with SciPy, a collection of Python modules for scientific computing which provides functions similar to those of MatLab. The currently available statistics are: sum or average duration, number of occurrences, earliest or average start time, latest or average end time. More statistics can easily be added, but these should be selected in a dialogue with secondary users.

# 6 Deployment

When the configuration planner has generated a new configuration for a given goal (set of activities), it can now be deployed simply by updating the preprocessing XML file. The preprocessing module then automatically reconfigures

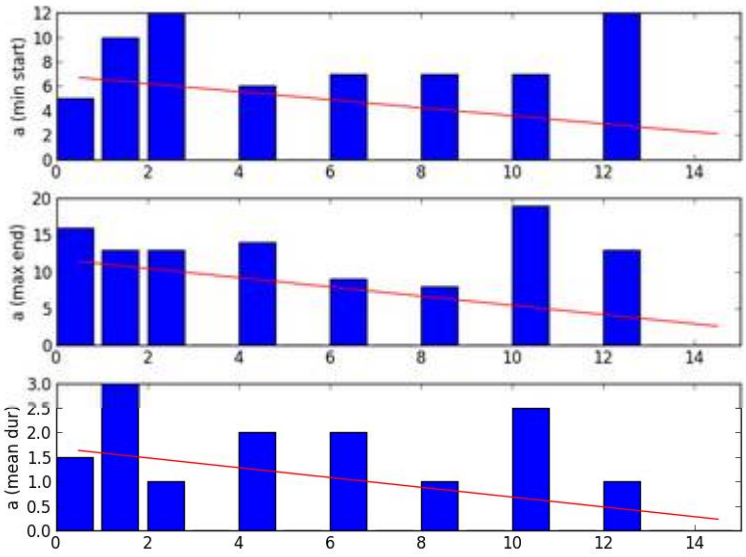


Figure 10: Three different statistics for a single activity.

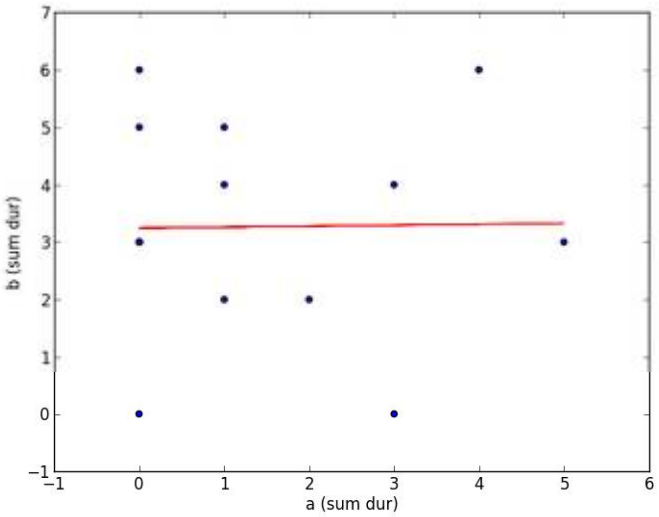


Figure 11: Scatter plot of statistics for two different activities. Each point represents a value for activity a (x axis) and activity b (y axis) in a given epoch. A regression line (red) is also included.

itself, and will request sensor data from the data base and execute the different functionalities/preprocessors specified in the XML file. See section 4 for details.

# References

[1] J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.

[2] Jorge A. Baier and Sheila A. McIlraith. Planning with preferences. *AI Magazine*, 29(4):25–36, 2008.

[3] Christophe Dousson and Pierre Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 324–329, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[4] S. Mckeever, J. Ye, L. Coyle, C. Bleakley, and S. Dobson. Activity recognition using temporal evidence theory. *Ambient Intelligence and Smart Environments*, 2(3):253–269, 2010.

[5] Réka Nagy, Mihai Alexandru Suciu, and D. Dumitrescu. Exploring lorenz dominance. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2012, Timisoara, Romania, September 26-29, 2012*, pages 254–259. IEEE Computer Society, 2012.

[6] 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*, 4(4):347–367, 2012.

[7] Arun K. Pujari, G Vijaya Kumari, and Abdul Sattar. Indu: An interval duration network. In *Proceedings of Sixteenth Australian joint conference on AI*, pages 291–303. SpringerVerlag, 2000.

[8] Lia Susana d.C. Silva-Lopez, Jonas Ullberg, and Lars Karlsson. On combining a context recognition system and a configuration planner for personalised ambient assisted living. In JuanCarlos Augusto, Reiner Wichert, Rem Collier, David Keyson, AlbertAli Salah, and Ah-Hwee Tan, editors, *Ambient Intelligence*, volume 8309 of *Lecture Notes in Computer Science*, pages 255–260. Springer International Publishing, 2013.

[9] Jonas Ullberg and Federico Pecora. Propagating constraints on sets of intervals. In *ICAPS Workshop on Planning and Scheduling with Timelines (PSTL)*, 2012.

# A Example of preprocessing XML file

The following XML file has been used for the second Swedish test site, and it consists of three parts. First, there are four preprocessors which take sensor data from the DB, for instance for `'Bed - Bedroom @ testsite_se_2'`, convert them to boolean values and store the result in time lines for state variables or activities such as `_in_bed`. Note that there might be multiple time lines for each state variables/activity, representing different interpretations due to the inherent uncertainty in the sensor data.

There is also a single rule for inferring the activity `_watching_tv` from two state variables `_tv_on` and `_in_tv_room`.

Finally, there are four extractors that selects the time lines with the maximal extent for each activity. These represent the end result of the context recognition process.

```
<rules xsi:noNamespaceSchemaLocation='rule_schema.xsd'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>

<preproc name='TunstallTrueFalse' in='Bed - Bedroom @ testsite_se_2'
out='_in_bed' args=''/>
<preproc name='TunstallPIRSimple' in='PIR - TV Room @ testsite_se_2'
out='_motion_bed' args=''/>

<preproc name='TunstallTrueFalse' in='TV - TV Room @ testsite_se_2'
out='_tv_on' args=''/>

<preproc name='TunstallPIRSimple' in='PIR - TV Room @ testsite_se_2'
out='_in_tv_room' args=''/>

<rule out="_watching_tv">
<constraint from="_watching_tv" type="during" args=""
to="_tv_on" />

<constraint from="_watching_tv" type="during" args=""
to="_in_tv_room" />
</rule>



<extractor name='max' in='_in_bed' out='in_bed' />
<extractor name='max' in='_motion_bed' out='motion_bed' />

<extractor name='max' in='_tv_on' out='tv_on' />



<extractor name='max' in='_watching_tv' out='watching_tv' />

</rules>
```

# B Pseudocode for configuration planning algorithm

In order to find solutions to configuration planning with partially-ordered preferences (C3PR) problems, algorithm 1 is here proposed. The algorithm searches the space of partial solutions, and is based on partial order planning. Algorithm 1 takes a C3PR problem consisting of a problem instance (goal, available functionalities, and domains of preference categories), a search heurstic and a dominance operator and returns a set of solutions.

The algorithm searches among configurations $\langle \mathbf{F}_i, Lk_i, Lk_c, \mathbf{Ex}, Pcat \rangle$ where $\mathbf{F}_i$ is a set of functionality instances, $Lk_i$ and $Lk_c$ are information and causal links between those instances, $\mathbf{Ex}$ are execution ordering constraints between the instances, and $Pcat$ is a vector with values for the different preference categories.

The solution to a C3PR problem is a set of dominating FAC, which hints at the importance of the dominance operator. Currently this algorithm supports solving C3PR$_{\mathcal{P}}$ (Pareto) and C3PR$_{\mathcal{L}}$ (Lorenz). Each dominance operator has its advantages. Solutions to C3PR$_{\mathcal{L}}$ are subsets of C3PR$_{\mathcal{P}}$, and our hypothesis is that being all other elements the same in a C3PR problem, a Lorenz Dominance operator will return a set of FAC that is at most as big as the Pareto set. To implement Lorenz dominance, we strictly followed the method proposed by Nagy et al in [5].

In order to compactly represent the domains $\mathbf{U}_{pref}$ of the preference categories in the implementation used for testing the algorithm, a semiring relating the semirings of the preferences was constructed.

For solving the C3PR problem given, the algorithm starts by using one heuristic focused on finding a first fully admissible configuration (FAC), and then switches to using preferences for scoring and comparing configurations. The algorithm keeps a search array with partial configurations, and an array of dominant fully admissible configurations (FAC). The array of dominant configurations contains the set of FACs that do not dominate each other, yet dominate every other FAC.

Once the first FAC has been found, a one-time event (algorithm 2) is triggered in which the heuristic value is recalculated into a preference-based score for every element in the search array. For every FAC found (including the first one) a pruning is triggered both in the search and in the dominant FAC array. All configurations dominated by the most recently found FAC are removed from both the search array and the array of dominant FAC.

**Algorithm 1** Algorithm for solving C3PR problems

**Input:** C3PR problem $P_{C3PR}$, first heur. $H_1$, dominance heur. $H_d$.
**Output:** Solution to C3PR problem $\mathbf{S}_{P_{C3PR}}$.

1. Let the goal functionality instance be $f_g = \langle \pi_{I_g}, \pi_{C_g}, \emptyset, \emptyset \rangle$, with the goal sentences as requirements.
2. Let the first configuration be $C_i = \langle \mathbf{F}_i, \emptyset, \emptyset, \mathbf{Ex}, Pcat \rangle$, in which $\mathbf{F}_i$ contains only $f_g$.
3. Let the search front array $S$ contain only $N_i = \langle C_i, h_i \rangle$, in which $h_i$ is the heuristic score of configuration $C_i$ according to $H_1$. $S$ will be sorted according to the heuristic values.
4. Let the solution array $\mathbf{S}_{P_{C3PR}}$ be empty.
5. If search front $S$ is empty, quit and return $\mathbf{S}_{P_{C3PR}}$.
6. Get and remove the first node $N_x$ from search front $S$.
7. If an effect of any $f_x \in \mathbf{F}_x$ of configuration $C_x$ in node $N_x$ threatens any link returned by $Lk_c$ of $C_x$ :
   (a) generate child partial configurations from conflict resolution.
   (b) check children for consistency.
   (c) send all consistent children to $S$.
   (d) Go to 5
8. If no unsatisfied requirements in $C_x$:
   (a) For each effect triggering a conditional preference in any $f_x \in \mathbf{F}_x$ of configuration $C_x$:
      i. Create a child configuration $C_{x1}$
      ii. Create an assignment in $Lk_c$ of $C_{x1}$ relating the triggering effect and the condition for the conditional preference.
      iii. Add a sequential execution constraint to $Ex$ of $C_{x1}$ reflecting the assignment.
      iv. If consistent, calculate heuristic value and add to $S$.
      v. After the last child was checked, go to 5.
   (b) If no conditional preferences are triggered:
      i. Call $proc(N_x, S, \mathbf{S}_{P_{C3PR}}, H_d)$
      ii. Go to 5.
9. Select unsat req $r_x$ of some $f_x \in \mathbf{F}_x$ of configuration $C_x$ in node $N_x$

(Continued.)

**Algorithm 1** Continued: Algorithm for solving C3PR problems

10. If $r_x$ is a causal requirement of $f_x$:
    (a) Create a child configuration $C_{x1}$
    (b) Instantiate a $f'$ with $r_{x1}$ as effect, and add $f'$ to $\mathbf{F}_{x1}$ if not already included.
    (c) Create an assignment in $Lk_c$ of $C_{x1}$ relating $f'$, $f_{x1}$ and $r_{x1}$.
    (d) Constrain $f'$ in $C_{x1}$ to come before $f_{x1}$, after the initial conditions, and before $f_g$.
    (e) Check $C_{x1}$ for consistency.
    (f) If consistent, calculate heuristic value and send to $S$.
    (g) Go to 10a until all possible ways to satisfy $r_x$ are checked.
11. If $g_x$ is an information requirement of $f_x$:
    (a) Create a child configuration $C_{x1}$
    (b) Instantiate a $f'$ with $r_{x1}$ as output, and add $f'$ to $\mathbf{F}_{x1}$ if not already included.
    (c) Create an assignment in $Lk_i$ of $C_{x1}$ relating $f'$, $f_{x1}$ and $r_{x1}$.
    (d) Constrain $f'$ in $C_{x1}$ to simult. with $f_{x1}$, after the initial conditions, and before or simult. with $f_g$.
    (e) Check $C_{x1}$ for consistency.
    (f) If consistent, calculate heuristic value and send to $S$.
    (g) Go to 11a until all possible ways to satisfy $r_x$ are checked.
12. Go to 5.

**Algorithm 2** Procedure $proc(N_x, S, \mathbf{S}_{P_{C3PR}}, H_d)$

**Input:** Search node $N_x$, Ordered Array $S$, Array C3PR $\mathbf{S}_{P_{C3PR}}$, dominance heur. $H_d$ (monotonically decreasing)

1. If $\mathbf{S}_{P_{C3PR}}$ is empty:
   (a) Recalculate heuristic value for $N_x$ using $H_d$.
   (b) Add $N_x$ to $\mathbf{S}_{P_{C3PR}}$.
   (c) Recalculate heuristic value for all elements in $S$ using $H_d$, pruning away those dominated by $N_x$.
   (d) Sort $S$.
2. If $\mathbf{S}_{P_{C3PR}}$ is not empty:
   (a) Prune from $S$ all nodes dominated by heuristic value of $N_x$.
   (b) Discard $N_x$ if dominated by any element in $\mathbf{S}_{P_{C3PR}}$.
   (c) Prune from $\mathbf{S}_{P_{C3PR}}$ all nodes dominated by heuristic value of $N_x$.
3. Go to 6.