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

D3.4 Final integrated context inference and configuration planning system

WP related to the Deliverable:	3
Nature:	P
Dissemination Level :	PU
Version:	V1.0
Author(s):	Lars Karlsson
Project Participant(s) Contributing:	ORU
Contractual Date of Delivery:	20141231
Actual Date of Delivery:	20150113

Document History

Version	Date	Type of editing	Editorial
0.1	01/12/14	Table of Contents	ORU
0.2	21/12/14	First complete draft	ORU
1.0	11/01/15	Final version	ORU

Deliverable Summary

This deliverable contains material that supports the delivery of the Final integrated context inference and configuration planning system. The latest version of the context recognition module has been operational in the test sites for over a year. There is a 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 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.

Finally, a panel for computing and displaying statistics for long-term trend analysis has been integrated with the DVPIS (developed in WP4). This supports the computation of a number of statistics for activities, and can for comparative purposes display both several activities and several statistics in the same graph.

Table of Contents

1 Introduction

The aim of WP3 is to provide context recognition services to GiraffPlus by inferring activities and other context information from the sensors distributed in the primary users' apartments. This is achieved as follows:

- The central component of is a *context recognition service* (task 3.1 according to the DoW) which takes sensor data from the data storage (developed in WP2), and infers activities for those upon requests made through the DVPIS (DVPIS stands for data visualization and personalization; see D4.3 for details).
- The context recognition service is supported by a *configuration planning service*, which determines how to obtain the sensor data needed by the former (task 3.2).
- The two services are integrated by allowing the configuration planning service to reconfigure the preprocessing module of the context recognition service (task 3.3).
- In order to support a *long-term analysis* of activities, statistics can be aggregated for inferred activities and displayed in the DVPIS (task 3.4). These statistics functions are part of the DVPIS, and are displayed in a designated panel.
- The context recognition service is deployed on a central server, and not in the individual apartments (task 3.5). Reconfiguration is done by reconfiguring the preprocessing module of the preprocessing module of the context recognition service.

1.1 Scope of document

This deliverable presents advances in WP3 during M30-M36 (the previous deliverable covered work done up to M30). The deliverable comes attached with two scientific articles that represent the cumulative work done in the work package which includes the last work done in this period. The two articles have been submitted to journals in the field of artificial intelligence during this period and are currently under review. One of the articles addresses context recognition (task 3.1.) and the other one configuration planning (task 3.2). The deliverable also describes progress in the other tasks (3.3-3.5).

1.2 Deliverable structure

The rest of the deliverable is organized according to the tasks in WP3. Section 2 reports on context recognition, section 3 on configuration planning, section 4 on integration between the former and the latter, section 5 on long term trend analysis, and section 6 on deployment. Section 6 discusses how the additional sensors presented in D2.4 might be used for context recognition, and 7 presents conclusions and future work.

1.3 Deviations from plan

There have been no significant deviations from the project plan.

2 Context Recognition

The work in context recognition has during the last period been focused on consolidation and publication. In particular, the context recognition system has been presented in an article that is

currently under review for *International Journal on Artificial Intelligence Tool*s (Ullberg, Loutfi, & Pecora, Submitted). The article is included as Appendix A, and we refer to it for details. The major contributions are:

- A system architecture consisting of: a preprocessing module where it can be specified (manually or by the configuration planning service) what sensors to use and how to process their data; an inference module which can infer context from sensor data given inference rules; and an extraction module that can select among multiple interpretations.
- Capability of the inference module to produce multiple interpretations of sensor data in order to make inferences more robust when samples might be missing or wrong.
- A query language for supporting queries by users.
- An experimental evaluation of the scalability of the inference algorithm, which indicates a linear complexity for sparse networks and a quadratic complexity for dense networks.
- A test case at one of the Swedish test sites involving a primary and a secondary user.

The context recognition server has been operational for more than a year now (on a server residing at the facilities of XLAB), and can be queried from the DVPIS developed in WP4 (see D4.3, section 4).

In the test case mentioned above, the secondary user (caregiver) reported that the context recognition service provided useful and sufficiently accurate information about the primary user's (resident's) activities. In particular, it was discovered that the primary user quite often went up during the night. The accuracy of this was verified by querying the primary user. We have also investigated the situation at test sites in Spain and Italy, by querying two secondary users (physicians). It appears that while certain activities were reliably inferred, others were not. In some cases, this can be explained by problems with sensors (e.g. an electrical usage sensor that was not working properly), but in other cases it might be the rules or the available data that are insufficient. Overall, the context recognition service is able to provide relevant information about activities, but in its current form of the service its usability is limited. "Improvements in the input language for specifying the temporal rules (which lies somewhat outside the scope of GiraffPlus), the sensor suite and/or the user friendliness of the service would be required before commercial deployment is possible. Training of secondary users should also be considered. These are issues that would need more research.

Besides providing activity data to secondary users, the context recognition service has also been used to provide input to a fall detection system, together with data from a wearable fall sensor. This has been published in a journal article (Koshmak, Lindén, & Loutfi, 2014).

3 Configuration planning

Regarding the configuration planning service, the major aim of the last 6 months has been to further publish our results. The configuration planning algorithm of GiraffPlus has been presented in an article that is currently under review for *Kunstlige Intelligenz* (Silva-Lopez L. S., Broxvall, Loutfi, & Karlsson, Submitted). The article is included as Appendix B, and we refer to it for details. The major contributions are:

- A domain representation for configuration planning which supports multiple categories of preferences on the selection of different components (typically sensors or programs for preprocessing sensor data) and how these components are combined. These categories, which are provided by the user, may include for instance power consumption, noise level, and reliability.

- An algorithm which find valid configurations while performing multi-objective optimization over preference categories, using Pareto and Lorenz dominance criteria.
- An extensive experimental evaluation of the algorithm, where 9 different parameters (e.g. number of available components and average number of requirements for each component) were varied in order to create 2200 problem families, and for each family 14000 problem instances were randomly generated and solved. The problems were in general solved in a matter of milliseconds or seconds.

We believe that the inclusion of preferences, besides opening up interesting scientific issues, also can contribute to make the system more robust and facilitate use acceptance. The representation of preferences is presently quite limited, and extending it constitutes important future work.
In addition, an article about an earlier version of the planner, without preferences, is still under review, for *Journal of Ambient Intelligence and Smart Environments (JAISE)* (Silva-Lopez L. S., Broxvall, Loutfi, & Karlsson, Submitted).

4 Integration of context inference and configuration planning

In the integration (which was first described in D3.3), we take advantage of the preprocessing module of the context recognition component, which can request sensor data from the data storage, preprocess this data in various ways (e.g. thresholding, aggregating) and then make it available to the inference module. The configuration of the preprocessing module is specified in an XML file. Hence, the configuration planner outputs configurations as XML-files which are read by the context recognition.
Figure 1 gives a schematic of the integration, including the various documents involved and the connections to the DVPIS (see D4.3) and data storage. Notice that it is the DVPIS (and in the extension the users) that decides what activities to look for in the data, and this in turn provides the goals for the configuration planner. The planning domain consists of a mapping between sensors and state variables, and a specification of functionalities available in the preprocessing module of the context recognition.

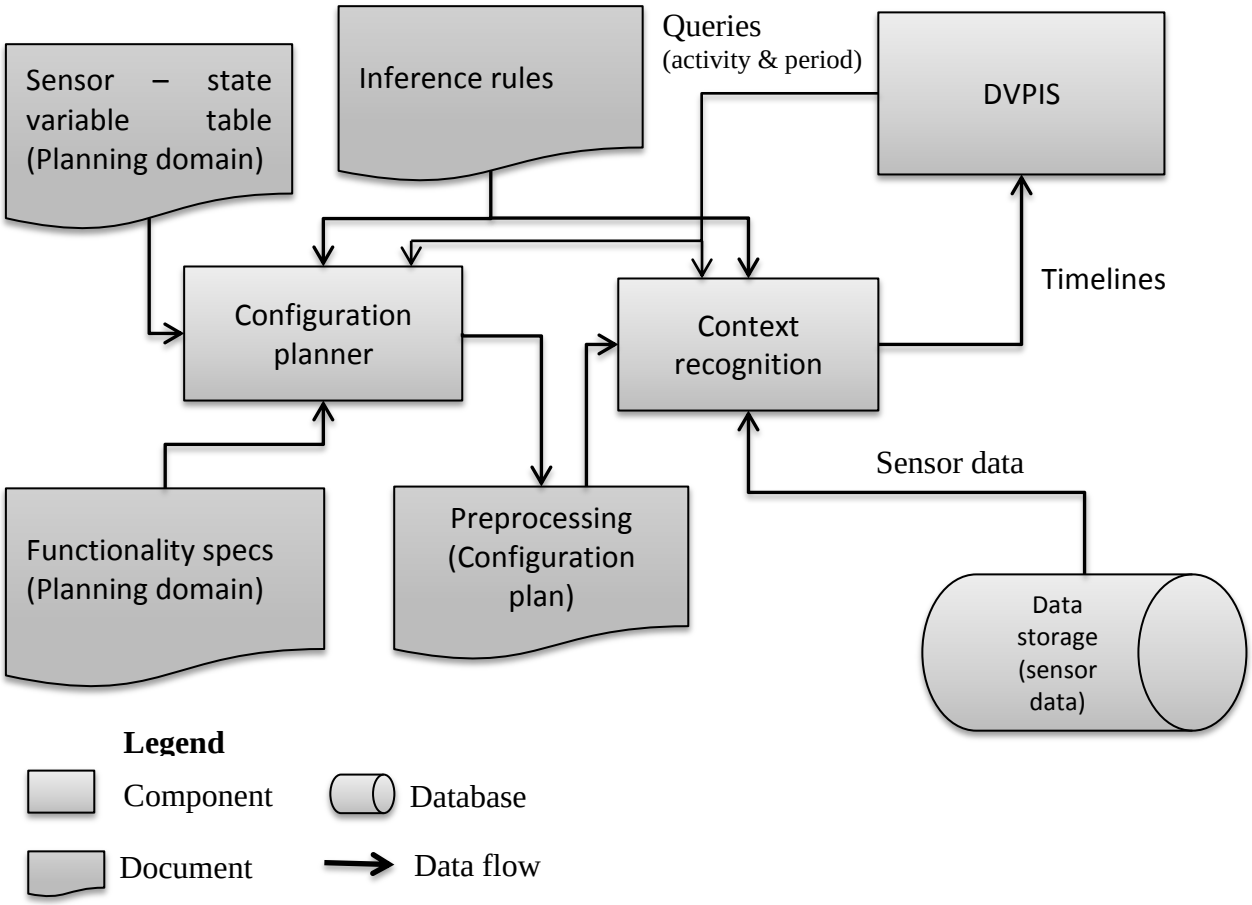


Figure 1 Schematic of integration of configuration planning and context recognition, with data flows, documents, and other components involved.

5 Long-term trend analysis

The major novel contribution during M30-36 has been the reimplementation and integration of the long term trend analysis (LTTA) into the DVPIS@Office, which is the client available for secondary users (physicians, caregivers etc). The purpose of the LTTA is to compute and display various statistics of activities, in order to discern patterns and trends over longer time. It uses the same data (sets of intervals for activities obtained from the context recognition service) as the activity viewer (see D4.3 section 4) but performs additional processing.

The LTTA is based on dividing the timeline into epochs, which are typically a day each. (Some of the statistics are mostly meaningful within day-long epochs). These day-long epochs can either be from midnight to midnight (day epochs), or from noon to noon (night epochs). The latter is useful for investigating night-time activities: if e.g. the primary user sleeps from 9pm to 7am, that should fit into one epoch. The different statistics that are currently available concern both how much/often an activity is performed, and when it is performed:

- Number of occurrences per epoch
- Sum of durations per epoch
- Average duration per epoch
- Earliest starting time per epoch

- Latest ending time per epoch
- Average time for activity per epoch

The LTTA can also compute running averages over several epochs. Typically, the longer the total period investigated, the more epochs should be used for computing this average. The running averages are important for separating the signal from the noise when a large number of epochs are considered. We preferred to use running averages to linear trends, as the former can capture more patterns than long-term increases and decreases of a parameter.

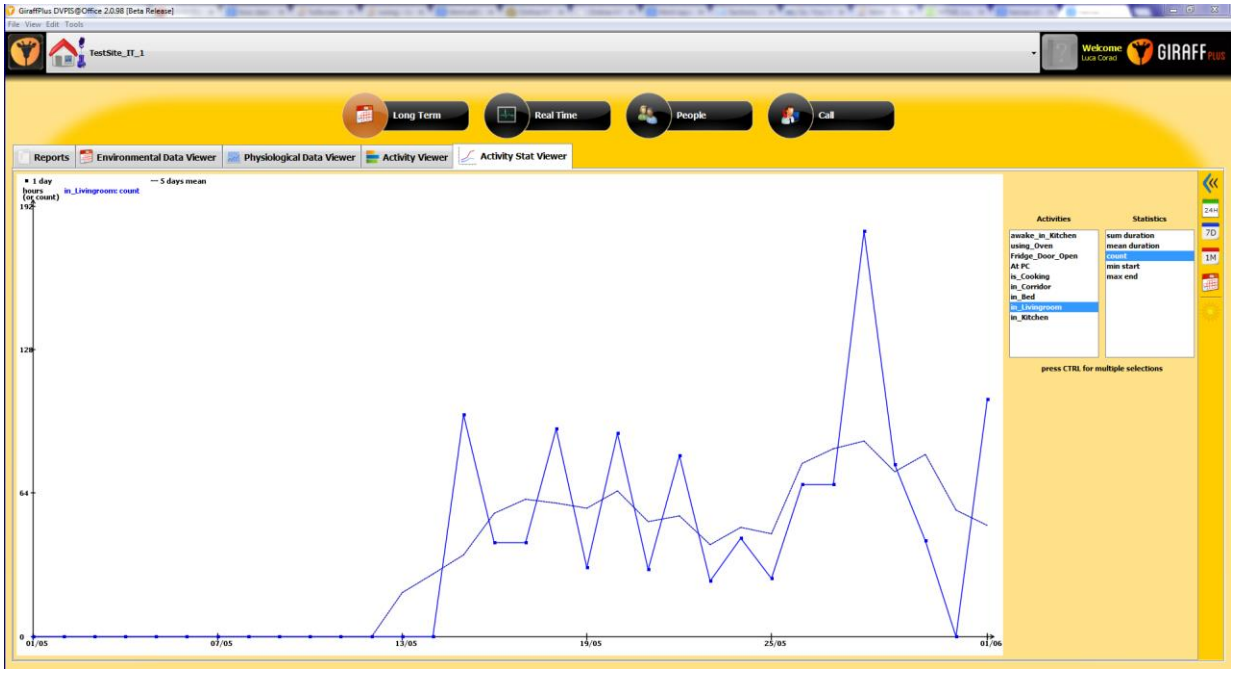


Figure 2 Example of LTTA panel opened in the DVPIS, showing one statistics for one activity (number of times the person is in the living room) over a month-long period. The user selects activities and statistics in the list boxes to the right, and period and epochs. There is also a smoothed curve (in darker colour) for the activity. The smoothing is done with a 5-day running mean. Data comes from one of the Italian test sites.

Figure 2 shows a window with LTTA opened from the DVPIS@Office application. The secondary user can select one or more activities, one or more statistics to apply to these activities, and what time and epochs (day or night) to use. This allows the user to both look at several statistics for the same activity at once, or to compare a statistic for several different activities, or even a combination of both. For periods longer than 10 days, the LTTA module also computes a smoothed version of the curve, using running averages. This is displayed as a dotted curve in a slightly darker colour.
Figure 3 shows the same period, but with night epochs instead.

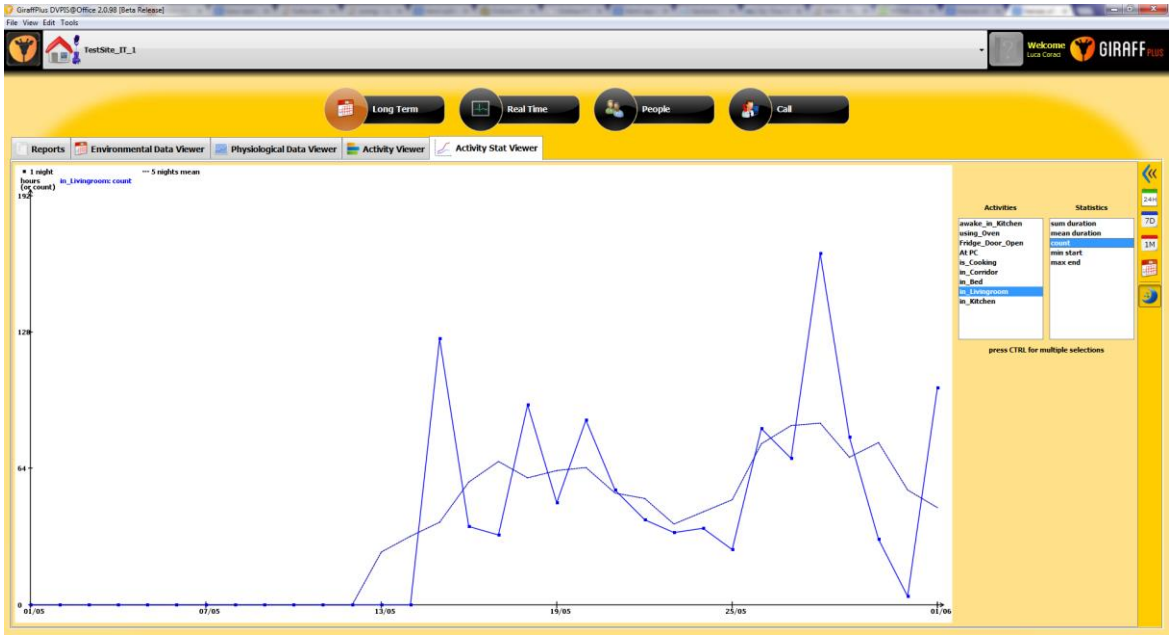


Figure 3 This panel displays the same activity, statistics and period as in Figure 2, but with epochs set to nights as indicated by the moon icon to the right.

Finally, Figure 4 shows a month with three activities and two statistics selected. Colours are assigned automatically to the different curves.

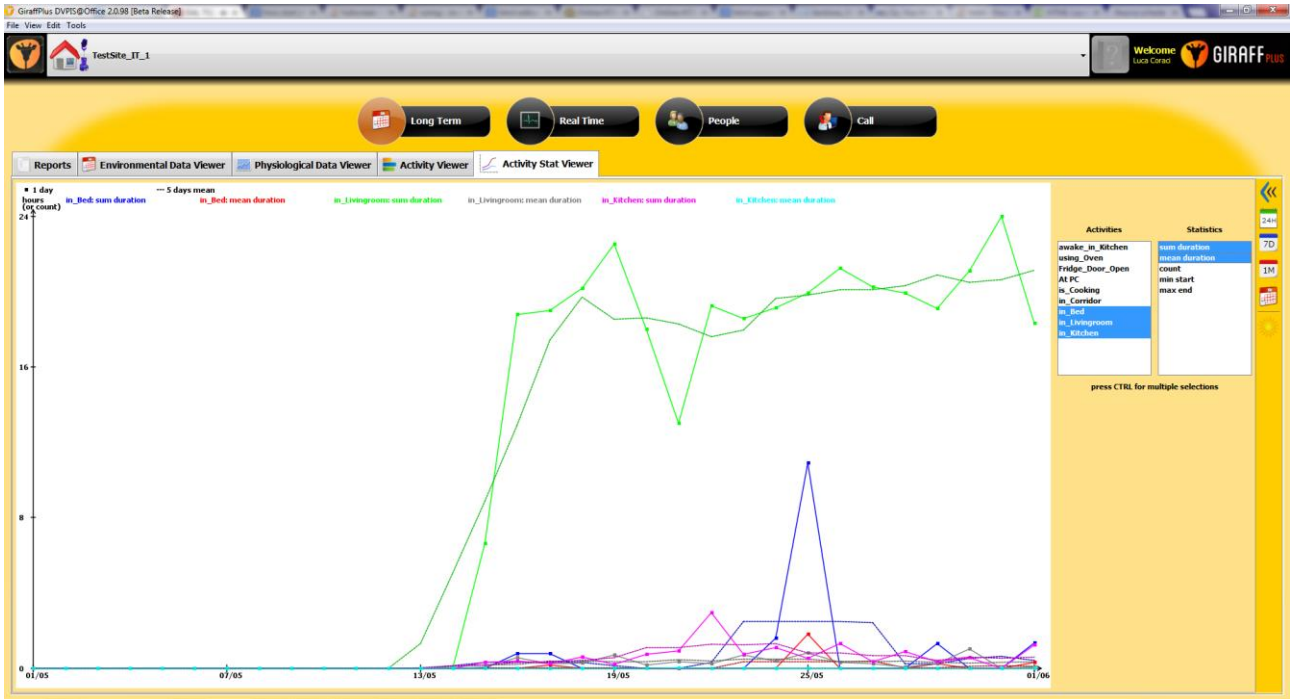


Figure 4 This panel displays three activities (*in bed*, *in living room* and *in kitchen*) and two statistics (sum of durations and average duration) over a month, including smoothed curves (in darker colours) for each combination. The smoothing is done with a 5-day running mean. One can see here that the resident spends most of the time in the living room (green curve).

6 Deployment

The context recognition is a Representational State Transfer (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. Such queries can be made by users through the DVPIS, in the activity and long term trend analysis views. The context recognition in turn gets sensor data from the data storage. The context recognition can also send alarm messages to Pushover clients. More details can be found in Appendix A.
Although we have verified that the configuration planning service can output valid configurations to the context recognition service, the former has not been used in the 15 GiraffPlus systems deployed with real users. As the entire GiraffPlus system is evolving, it has been more convenient for our engineers to specify configurations manually. In addition, the current sensor suite is fairly small and has not much redundancy, so there is in practice not much work for a configuration planner to do. However, we believe that in a more large-scale and long-term deployment of a more stable version of the system, the configuration planner would provide valuable services and in particular increase the robustness and adaptability of the system.

7 Conclusions and future work

The main effort during M30-36 has been on publishing the scientific results produced in the work package. Presently, three journal papers are under review (Ullberg, Loutfi, & Pecora, Submitted) (Silva-Lopez L. S., Broxvall, Loutfi, & Karlsson, Submitted) (Silva-Lopez L. S., Broxvall, Loutfi, & Karlsson, Submitted). These submissions mainly concern tasks 3.1 (Context recognition) and 3.2 (configuration planning).
The main contribution to the GiraffPlus system during M30-36 is the long term trend analysis (task 3.4), which has been integrated into the DVPIS. For the other tasks, including integration (3.3) and deployment (3.5), the implementation work was by large completed at M30. The context recognition service, which is the most important contribution of WP3 to the GiraffPlus system, is currently running as part of the system. The configuration planning service has also been integrated, but is currently not actively used.

There are many opportunities for future work. During the last months of the project, some novel sensors have been investigated within WP2: a collection of ZWave sensors (alarm buttons, PIR temperature/light, door/window contact, electrical/lamp socket etc); a Samsung Gear Live watch (heart rate meter, accelerometer, gyroscope, compass, step counter); and a Garmin cardio chest strap. In order to take advantage of these new sensors in the context recognition, it would sufficient to have their data stored in the data storage, and possibly also develop preprocessing components for them. They could then be used by the context recognition and configuration planning components by including them in the relevant documents according to Figure 1.
In order to be part of a commercially deployable system, the context recognition service would need more systematic evaluations and subsequent improvements. If the numbers and types of sensors are extended, for instance with the ZWave sensors mentioned above, this could also give more opportunities for the configuration planning service. As mentioned in D3.3, there is also potential to develop a more systematic approach to the context recognition preprocessing by

introducing 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.

References

Koshmak, G., Lindén, M., & Loutfi, A. (2014). Dynamic Bayesian Networks for Context-Aware Fall Risk Assessment. *Sensors*(14), 9330-9348.

Silva-Lopez, L. S., Broxvall, M., Loutfi, A., & Karlsson, L. (Submitted). A partial-order configuration planner with local information heuristics for generating configurations of devices in robotic ecologies. *Journal of Ambient Intelligence and Smart Environments*.

Silva-Lopez, L. S., Broxvall, M., Loutfi, A., & Karlsson, L. (Submitted). Towards Configuration Planning with partially ordered preferences: representation and results. *Kunstlige Intelligenz*.

Ullberg, J., Loutfi, A., & Pecora, F. (Submitted). A Context Recognition Toolset for Elder Care. *International Journal on Artificial Intelligence Tools*.

Appendix A

International Journal on Artificial Intelligence Tools

A Context Recognition Toolset for Elder Care*

Jonas Ullberg
Center for Applied Autonomous Sensor Systems, Örebro University
SE-70182 Örebro, Sweden
jonas.ullberg@oru.se

Amy Loutfi
Center for Applied Autonomous Sensor Systems, Örebro University
SE-70182 Örebro, Sweden
amy.loutfi@oru.se

Federico Pecora
Center for Applied Autonomous Sensor Systems, Örebro University
SE-70182 Örebro, Sweden
federico.pecora@oru.se

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

This paper provides an overview of a complete context recognition toolchain that has been developed for the GiraffPlus project which performs activity monitoring in the homes of elderly. The paper discusses the details of the context recognition system itself, both from an algorithmic and semantic point of view, and describes how it is integrated with other services (such as alarms and visualization software). The experimental validation presented in this paper evaluates the performance of the algorithm and the practical applicability of the system in terms of forming queries that are relevant for the users. The system has been tested and deployed in fifteen real European homes inhabited by elderly people with varying levels of mobility and healthiness.

Keywords: Context Recognition; Temporal Constraints; Elderly Care

1. Introduction

Enabling elderly people to live independently in their own home after the onset of ageing related health problems is beneficial both for the society and the individual. From the individual's point of view, staying at home is positive since it allows them to keep established habits and comforts. From a societal point of view this is also beneficial since home care is often more cost-effective than nursing home care.

*This work was supported by the European Commission in the framework of the GiraffPlus FP7 project (Contract no. 288173).

The current rapid increase in ageing population growth makes the latter argument increasingly important.

In order to further facilitate independent living among the elderly, the relative benefits of nursing home care need to be brought into homes. In Sweden, for instance, home care consists of scheduled visits by caregivers, complemented with wearable alarm bracelets that the elderly can use to alert the caregivers in case an accident occurs (e.g., a fall). This situation can be improved upon by introducing new, non-intrusive, technology into their homes that allows caregivers to monitor the habits of the elderly in order to proactively detect signs of possible health deterioration, and alert caregivers of dangerous situations when the elderly is unable to do so on their own.

Thus, a key enabler for independent living is automated behaviour monitoring coupled with pervasive sensors which can provide the mentioned capabilities. In particular, such a system should possess several key qualities: (**requirement 1**) it must allow physicians to perform "activity queries" on different aspects of the elderly person's daily life based upon known or suspected medical conditions; (**requirement 2**) it should be possible to trigger notifications should certain activity patterns or hazardous situations emerge from the sensor data; (**requirement 3**) the behaviour monitoring system should be capable of performing queries over long horizons, as it is often important from the caregiver's point of view to measure trends over weeks and months; and (**requirement 4**) the system should be easy to deploy by caregivers with little or no technical training, and should therefore not be brittle to small changes in sensor placement and imprecise sensor readings.

As we explain in the evaluation section of this article, the above requirements emerge from real experience with elderly people and their caregivers. The focus of this article is the Context Recognition (CR) system — specifically, the knowledge representation and reasoning techniques developed to address the requirements above. We discuss two key innovations of our approach, namely a novel representation of sensor readings which captures the uncertainty deriving from coarse sensor placement and noisy readings, and the ability to reason upon long periods of time. Data from the environmental sensors are processed by the CR system through qualitative temporal models that define conditions on how activities are inferred. These models used are specified in the form of temporal relations among sensor readings. The models can be defined, added and/or removed dynamically to a list of *behaviours* that are specified by caregivers, and which is alterable online.

The presented CR system has been developed as a part of a larger system called GiraffPlus [4] which is developed as an EU-FP7 funded project. The GiraffPlus system includes a network of sensors placed in the home or worn by the elderly and is deployed in fifteen homes in three different countries, namely Sweden, Spain and Italy. These include physiological sensors such as weight and blood pressure, as well as environmental sensors like motion, pressure, temperature, and electrical usage sensors. The GiraffPlus system is named after one of its components: the Giraff telepresence robot. The robot uses a Skype-like interface to allow caregivers

to virtually visit an elderly person in the home. The Giraff telepresence robot is used as the primary means of communicating with the elderly; results of trend analysis, details of the models used for monitoring, and currently active sensors can all be accessed and/or manipulated on the Giraff robot's interface. Physiological sensors and the integration of the telepresence robot lie outside the scope of this article and we refer the interested reader to [4, 22] for details of the complete GiraffPlus system.

The contributions of this paper include (1) a CR system that accounts for models of human behaviors that are modeled from first principles by caregivers (in adherence to **requirement 1**), (2) a novel, efficient constraint-based context inference engine that accounts for uncertainty in sensor readings (in adherence to **requirements 3** and **4**), and (3) a system for continuous automatic inference of behaviors of interest (in adherence to **requirement 2**). Benchmarks and real-world scenarios are used to validate the proposed system. The latter consist of a test case with an elderly Swedish user in Örebro.

The paper is organized as follows. Section 2 provides the necessary background. Section 3 describes the specifics of the constraint propagation algorithm that is used by the CR system. Section 4 describes how the entire GiraffPlus system and its services are implemented. Sections 5 and 6 describe the CR service in greater detail and the modeling language provided to caregivers for specifying the behaviors that are to be monitored. Finally, Section 7 provide an evaluation of the CR from a user perspective.

2. Background

Current approaches to the problem of recognizing human activities can be roughly categorized as *data-driven* or *model-driven*. In data-driven approaches, models of human behavior are acquired from large volumes of data over time. Notable examples of this approach employ Dynamic Bayesian network (DBNs) in conjunction with learning techniques for inferring transition probabilities [23, 35]. Extensions of these approaches have been proposed for dealing with realistic features of the domain, such as interleaved activities [9, 21] and multiple persons [29].

Although highly effective in specific domains, such systems are typically brittle to changes in the nature and quantity of sensors, requiring significant re-training when the application context changes. This contrasts with the requirement that the criteria for CR can be specified on-line depending on circumstances assessed by secondary users and put to service immediately, without the need for re-training and model tuning (**requirement 1**). Liao et al. [18] have described an approach which partially overcomes these limitations using conditional random fields, showing that learned behavior models can be generalized to different users. However, this has been empirically proved only for the specific context of activity recognition using GPS traces and location information, and does not cater to user defined queries that can be posted online. A complementary approach is followed by Helaoui et al. [14] to overcome some of the limitations of purely data-driven techniques. Specifically,

the authors incorporate modeling capabilities to capture features such as qualitative temporal relations which describe how events relate to each other. One of the key features of the approach is its capability to recognize interleaved activities. However, it is limited to detecting sensor context, and the applicability of the approach to a highly-dynamic context, like our use case, is untested. Nor do any of the approaches above address the issue of triggering notifications or measuring trends over long periods of time (**requirements 2** and **4**).

Model-driven approaches to activity recognition follow a complementary strategy, in which patterns of observations are modeled from first principles rather than learned or inferred from large quantities of data. Such approaches typically employ an abductive process, whereby sensor data is explained by hypothesizing the occurrence of specific human activities. Examples include work by Goultiaeva and Lespérance [11], where the Situation Calculus is used to specify very rich plans, as well as the work of Augusto and Nugent [3], Jakkula et al. [16], Pinhanez and Bobick [25], all of whom propose rich temporal representations to model the conditions under which patterns of human activities occur. As we show in this paper, the temporal relations between sensor readings and human activities are an essential element of modeling, and our system uses temporal relations as the primary representational element in modeling. Other techniques used to perform CR include ontological reasoning. For instance, Springer and Turhan [30] employ OWL-DL to specify models of complex situations, the argument being that the more complex the situation to recognize, the more sophisticated the behavior of the smart environment. However, time is considered only implicitly, and the ability of the system to infer context online over long horizons, as well as its ability to cope with imprecise readings, is not established (**requirements 1**, **3** and **4**). Similarly, Riboni and Bettini [27] combine ontological and statistical reasoning to reduce errors in context inference, albeit without addressing temporal relationships between activities.

Data- and model-driven approaches have complementary strengths and weaknesses: the former provide an effective way to recognize elementary activities from large amounts of continuous data — relying, however, on the availability of accurately annotated datasets for training; conversely, model-driven approaches provide a means to easily customize the system to different operational conditions and users through expressive modeling languages — which, though, is based on the ability of a domain modeler to identify criteria for recognition appropriately from first principles.

The CR engine presented in this work is related to temporal constraint-based approaches such as SAM [24] and constraint-based chronicle recognition [7]. 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 Mckeever et al. [20] underscores the advantage of explicitly accounting for activity durations. Our work introduces a key novelty in temporal constraint-based CR, 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.

3. Propagating Temporal Constraints on Sets of Intervals

In this work, context is recognized from sensory data that is represented as sets of intervals on timelines. The model that describes the causal relationships between the states of the sensors and the inferred activities is provided as a set of temporal constraints in Augmented Allen's Interval Algebra (AAIA). A relation in AAIA of (a) two intervals i_1, i_2, (b) a qualitative relation in Allen's Interval Algebra [AIA, see 2], and (c) a (possibly empty) set of bounds on the distances between relevant time points of the relation. Relevant time points of a relation in AIA are those which define the semantics of the relation, e.g., the semantics of the precedence relation i_1 `PRECEDES` i_2 is defined as the end time of i_1 being less than the start time of i_2. AAIA constraints allow to bound the distances between the relevant time points of the intervals involved in the relation, e.g., i_1 `PRECEDES`$\{[l, u]\}$ i_2, or i_1 `DURING`$\{[l_1, u_1], [l_2, u_2]\}$ i_2 — see Table 1. An interval represents a fact that holds true during a limited period of time, e.g., the fact that a person is at a particular location. These facts are then related to human activities with the help of AAIA constraints. For instance the constraint $Cooking$ `DURING`$\{[l, u]\}$ $InKitchen$, describes the relation between the activity "Cooking" and the fact "In Kitchen", that is, cooking occurs when the human is sensed as being in the kitchen, and the bounds l and u in the AAIA relation represent optional temporal "margins". Metric bounds on qualitative relations are useful, for instance, to distinguish between events that happened a minute ago and a week ago. Other approaches to context recognition [e.g., 14] employ only qualitative relations, and therefore have to make strict assumptions about the number of occurrences of an activity within a fixed temporal window.

In this work, both sensor traces and hypothesized human behaviors are represented as intervals. Assessing whether a model of human behavior holds given the sensor traces consists of attempting to constrain a hypothesized behavior with intervals representing sensor readings according to the constraints given in the model.

Each interval represents a fact about a *state variable*. State variables represents either sensors or behaviors of the human user. We indicate that an interval i represents a fact about state variable S as $var(i) = S$. The set of intervals referring to one state variable constitute a *timeline*, that is, a stepwise-constant function mapping time to values of a state variable.

A model of human behavior is a collection of tuples $\langle v, \{v_1\ r\ v_2 : r \in \text{AAIA}\}\rangle$, where v is a state variable representing human behavior, and v_1 and v_2 are state variables representing human beaviors or sensors. Each tuple expresses a "temporal rule", that is, a set of temporal relations that must hold between a particular

behavior v and sensor readings or other behaviors v_1 and v_2. For instance, the model corresponding to the example above is:

$$\begin{aligned}&\langle \mathit{Cooking}\\ &\{\mathit{Cooking}\ \texttt{DURING}\ \mathit{In\ Kitchen},\\ &\ \mathit{Cooking}\ \texttt{CONTAINS}\ \mathit{Stove\ On}\}\rangle\end{aligned}$$

Intervals representing sensor readings and inferred context are maintained in a network of AAIA constraints. A well known approach for verifying whether modeled human behavior is consistent with respect to sensor readings is to cast the problem as a *Simple Temporal Problem* [STP, see 6]. The time points in the STP represent the start and end times of the intervals, and the AAIA constraints in the model of human behavior are cast as simple distance constraints among these time points. It can be easily shown that a model of human behavior holds if and only if the resulting STP is consistent [7, 24]. In case of consistency, the admissible bounds of the time points in the STP represent the minimum and maximum times at which the time points can occur. The time points corresponding to the interval that represents the hypothesized behavior thus represent the temporal bounds within which the sensor readings indicate that the activity takes place. Overall, each tuple in the model can be considered a *query*: the example above represents a query by the caregiver which asks whether the user is cooking (iff the resulting constraint network is consistent), and when this activity occurs (indicated by the bounds of the interval on the *Cooking* timeline).

While a STP-based approach using such models allows for a convenient specification of relations, these expressions are brittle in that small deviations in raw sensory data can prevent certain activities from being inferred. To circumvent this issue, we propose here a method, first presented by Ullberg et al. [34], which allows many interpretations of sensor readings to be admitted. This is done by performing temporal inference on multiple intervals contextually, where each sensor reading is represented as a set of *flexible* temporal intervals rather than only one. To motivate the proposed approach, we provide an illustrative example which is inspired by deployed sensors in a real home.

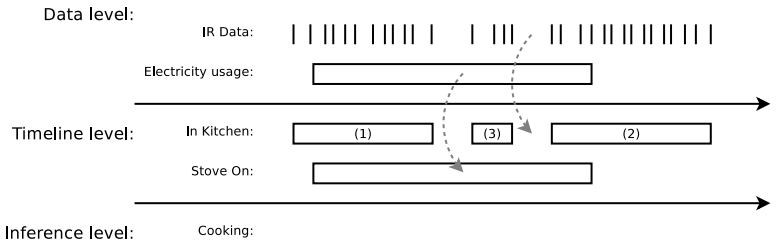


Figure 1. This Figure shows a situation where the activity *Cooking* can not be inferred due to the fact that there is no placement of it on the timeline where it is both `DURING` *Cooking* and `CONTAINS` *Stove*. Thus, after inference the *Cooking* timeline is empty.

Consider the situation illustrated in Figure 1. Unprocessed data coming from two sensors have been illustrated topmost in this figure: first, an infrared (IR) sensor that provides information about when motion is sensed in the kitchen; second, an electrical usage sensor that senses when the stove consumes electricity. For simplicity, we assume that the stove sensor is perfect and provides the ground truth. The IR data however needs to be interpreted to tell when the inhabitant is present at a location (in contrast to the sensing of motion which is what the data represents). In this example, the IR data is preprocessed in such a way that it forms three distinct intervals on the "In kitchen" timeline.

In this example, the model stated above for recognizing *Cooking* does not hold, as no *Cooking* interval can be placed on the corresponding timeline so that the two constraints hold with respect to the existing intervals on the *In Kitchen* and *Stove On* timelines.

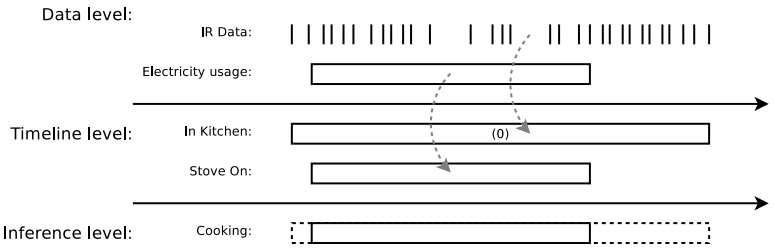


Figure 2. Unlike in the situation shown in Figure 1, here the *Cooking* activity can be inferred due to the fact that it can both be placed so that it occurs `DURING` *Cooking* and `CONTAINS` *Stove*. The dashed contour shows residual flexibility in the *Cooking* interval, i.e. any start and end time of the interval in question can be extended within its bounds, the solid interval has the minimal duration however.

Note, however, that Figure 1 represents an arbitrary interpretation of the IR sensor readings. This interpretation ignores the fact that the readings which lead to the separation between the intervals (1), (2) and (3) may simply have been due to a blind spot in the kitchen — recall, we must assume that sensor are placed approximately by the caregiver, and can be slightly moved over time. Figure 2 shows an alternative interpretationin in which *Cooking* is successfully recognized. Even though these two interpretations are visually similar, they are very different from the point of view of the constraint-based inference. Clearly, the rule is written with the scenario that unfolds in Figure 2 in mind, and small deviations from this "prototypical scenario" can prevent us from recognizing the activity. The problem illustrated in Figure 1 can be overcome by altering the rule so that *Cooking* is required to be `OVERLAPPED-BY` *In Kitchen* rather than occur `DURING` the same. This constraint would however not be satisfied if *In Kitchen* is first sensed after *Stove On* becomes true.

In general, one could argue that the above rule specification is simplistic, and does not take into account the multiple possible evolutions in time of the sensors

when the cooking activity occurs. However, forcing the modeler to cater for all possible evolutions of sensor traces has the obvious drawback of over-burdening the process of modeling the behaviors of interest: caregivers have a rough idea of "how" a cooking activity appears in terms of the sensors that have been placed in the environment — and qualitative temporal relations offer a convenient and intuitive means to capture this knowledge — but should not be required to foresee all possible situations. Note also that sensor placement may change slightly over extended periods of time (e.g., trivially, because the environment is cleaned and small displacements of sensors occur). These factors motivate the need for reasoning with *multiple interpretations* of sensor readings to enlarge the applicability of simple user-defined temporal queries.

We propose an approach that it is able to infer context from more relaxed interpretations of the sensor readings than what is typically possible, something that has proven to be an obstacle in the past due to the symbolic nature of the model. The core advantage is the ability to generalize the interpretation of sensory events over time, so that the interpretations of data take into account the timespan in which the query is grounded in. This is useful in the example above, where a more lax interpretation of the IR sensor allows to recognize cooking even if the given model is simple. Other practical use cases motivate the need for multiple interpretations of sensor readings. For instance, electrical usage sensors coupled to microwave ovens will provide different readings depending on whether the oven is used to heat a meal at 800 W or to defrost frozen goods at 90 W. Different interpretations can be used to ground brief but intense heatings of meals as separate events from longer defrosting instances at low power.

In our approach, we provide an algorithmic solution that employs multiple interpretations of sensor readings; the collection of these interpretations encompasses a wider range of possible intervals as support for the constraint-based context inference procedure. For instance, we might wish to use an interval (generated by a relaxed interpretation of the sensory data, biased towards generating large continuous intervals rather than introducing discontinuities) and all of its sub-intervals as support for the inference. The key issue hence becomes that of developing *temporal reasoning algorithms* that are able to reason on multiple interpretations of the same sensory data, using each of these interpretations as additional support for inferring an activity. In the following section, we provide a formal description of the representation and reasoning solutions we have developed for this purpose.

3.1. *Formal problem statement*

Our CR problem can be described as a Constraint Satisfaction Problem [CSP, see 33] of the form $\langle V, CA \rangle$. Here, $V = \{v_0, \ldots, v_l\}$ is a set of variables, each representing the timeline of a sensor or of one inferred activity. The domain of each variable, $v = \{i_0, \ldots, i_m\}$, is a set of (possibly overlapping) temporal intervals of the form $i = [s, e]$, where s is the start time of the interval and e its end time. Each such

interval either denotes that a sensed fact holds true, or that an activity was performed as the interval's time-span states (not during, but precisely starting at time s and ending at time e).

$CA = \{c_0, \ldots, c_n\}$ is a set of constraints on the variables in V of the form $c_i = \{(v_0\ r\ v_1)\}$, where each c_i constrains the domains of two variables. v_0 is a variable representing the timeline of an inferred activity, r is constraint in AAIA, and $v_{i\neq 0}$ is either a variable representing the timeline of a sensor or of another inferred activity. Note that this implies a dependency graph among timelines of inferred activities which has no loops, i.e., a tree. These constraints define temporal relations between the variables that should hold in order for an activity to be inferred. Table 1 shows all thirteen constraints in AAIA. This table does not show the quantitative aspect of the constraints, but rather the thirteen unique qualitative relations that a pair of intervals can have. Recall that the quantitative algebra we employ allows for the quantification of distances between the start and end times of the respective intervals, e.g. "a starts between 10 and 20 time units before b".

A solution to the problem $\langle V, CA\rangle$ is an assignment of values (i.e., sets of intervals) to variables (i.e., activity and sensor timelines). A solution to the CR problem is the projection of a solution to the CSP on the variable representing the inferred activity. In other words we are not interested in the interpretations of sensors readings necessary to support inferred activities.

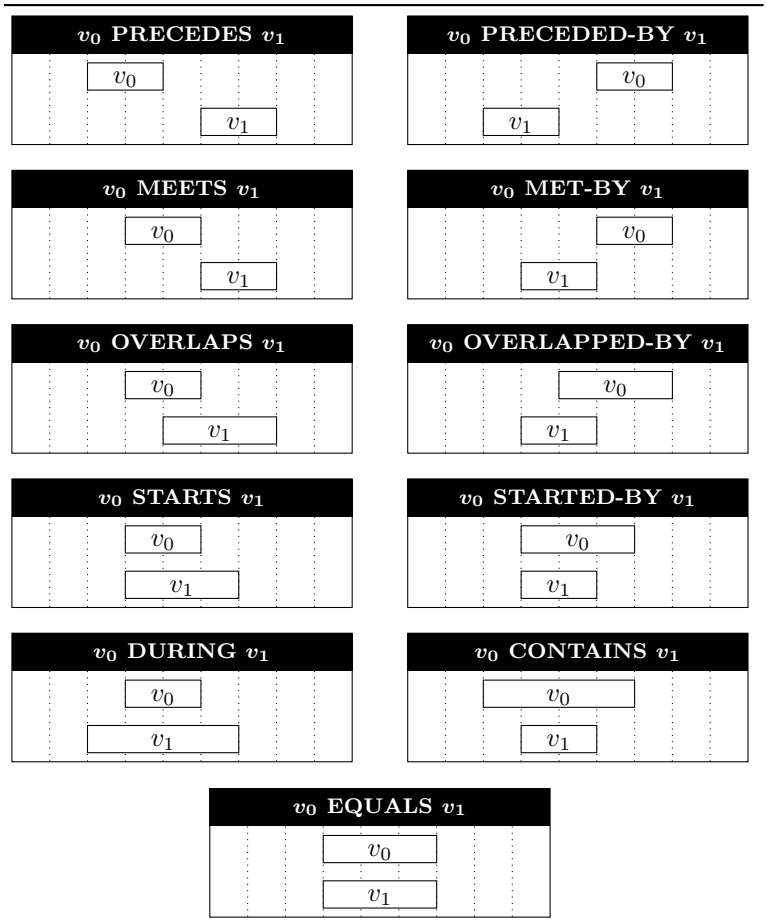
In the CSP, we maintain only one variable representing an activity to be inferred. The reason has to do with constraint propagation. Let an activity to be inferred be A. Propagating the constraints in the CSP may reduce the domain of a variable representing a sensor, S, which is necessary to support A. However, this reduction only reflects the fact that some intervals in the domain of S are not relevant for inferring A, and not that they represent incorrect knowledge about the sensor readings. The intervals filtered out due to the requirements of A could be used to infer another activity B. This is not possible if the CSP contains variables representing both A and B.

We employ a geometric representation of the domains of the variables in V. This representation allows us to define a propagation algorithm which achieves arc-consistency.

3.2. *Representing multiple intervals*

In order to increase the number of activities that we can successfully identify it makes sense to perform temporal inference on batches of interpretations contextually, or even more useful, on an entire spectrum of interpretations including these. In a naïve way, the former could be accomplished by admitting several overlapping intervals on the same timeline. For instance, by merging the interpretations in Figure 1 and Figure 2. This would however only work to a limited extent since it would also increase the complexity of searching for matching patterns in the data. This problem affects all approaches to CR which rely on an explicit representation of each interval

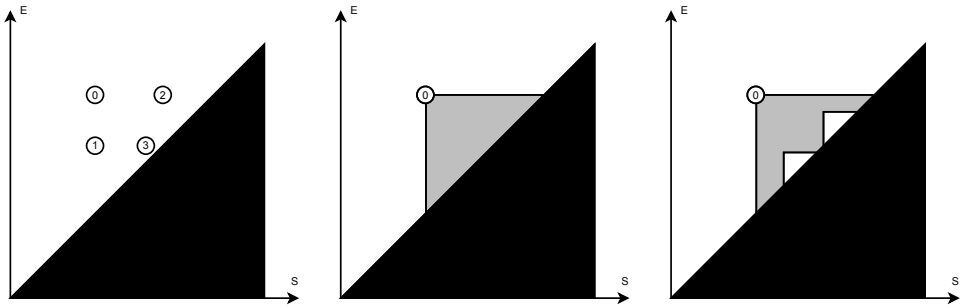
Table 1. The thirteen relations of Allen's Interval Algebra



Note: In our quantified version of Allen's Interval algebra we parameterize the admissible temporal distances between the start and end times, while doing so we relax the meaning of the algebra so that it is not always distinct or exhaustive — although such reasoning is possible. The rationale is that making a distinction between `PRECEDES` or `MEETS`, for instance, would have little meaning to the user since no two events occurs *exactly* at the same time.

in memory.

A more intelligent strategy is to propagate constraints on a spectrum of interpretations contextually. This requires changing the way in which we represent sets of intervals. The most straightforward way of representing a set of intervals would be to interpret an interval as a single point in a two-dimensional space as in Figure 3a. Each of the four points (intervals) in this space corresponds to one of the intervals in Figure 1 & 2. In this space, each point's projection onto the x-axis defines the interval's start-time, and its projection onto the y-axis the end time. Naturally, an



(a) A two-dimensional representation of a set of enumerated intervals. The numbered intervals (points) in the figure corresponds to the ones in Figure 1 and Figure 2.

(b) The set of intervals that occurs `DURING` interval 0 in Figure 3a.

(c) The set of intervals that occurs `DURING` interval 0 in Figure 3a, excluding all intervals contained within the two "gaps" in the timeline in Figure 1.

Figure 3. Set based representations of the timelines shown in Figure 1 and Figure 2.

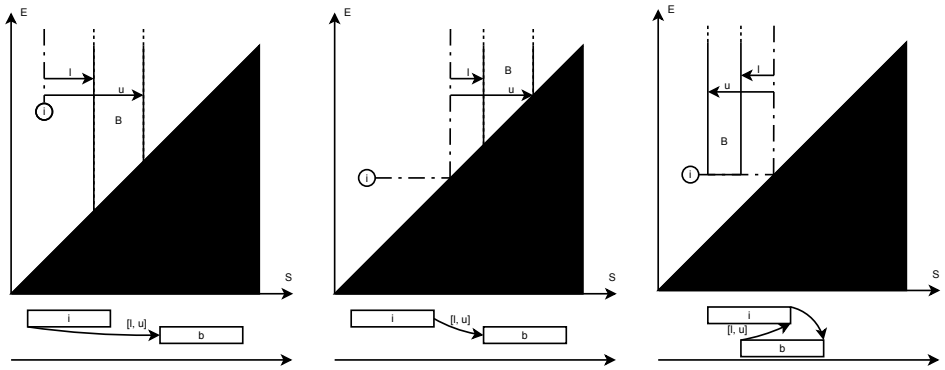
interval is not permitted to end before it has started, therefore no interval is allowed to reside in the lower-right part of this space. Interpreting intervals as points in a 2D space was first proposed by Rit [28], who named the representation Sets of Possible Occurrences (SOPOs), and described how qualitative Allen Interval constraints could be used and propagated on such representations. This representation was later discussed by Pujari et al. [26] and has subsequently only been briefly mentioned in other work [1, 8]. The reason for this lack of attention is most likely because of the introduction of alternative problem formulations such as the STP, TCSP [6] and DTP [31]. However, SOPOs cater exactly to the representational needs of our application, where we are not interested in representing disjunctions of constraints (as done in DTPs), and where a single interval representation (as provided by the STP) does not suffice. Furthermore, SOPOs allow to represent compactly large sets of intervals. Thus there is reason to believe that this representation is better suited for our particular problem.

SOPOs can be used to "generalize away" the usage of enumerated sets of intervals, considering instead groups of intervals. Figure 3b visualizes such a set of intervals. Specifically, the gray region protruding from the diagonal in this figure corresponds to the set of all intervals that are `CONTAIN`ed within Interval 0 in Figure 2. For an interval to be contained within another, the requirement is that the interval starts after and ends before the "containing" interval. These two requirements corresponds to the bounds of the gray area in the figure. Thus, this area contains all the intervals found in Figure 3a.

By looking at the figures we can also notice that it might be meaningful to reason about the set of intervals that are fully contained within Interval 0, with the exception of the sub-intervals that are contained in the two "gaps" in Figure 1, where we have no sensor data that can support an assumption of the person being

in the room. Figure 3c illustrates the mentioned set. The rationale behind this could for instance be that we want to be more general in our description of the state of the world and use facts such as "The person was in the kitchen between 13:00 and 14:00" (contained in interval 0, arbitrary picked time not illustrated in any figure) or "between 13:00 and 13:15" (contained in interval 1), but not "between 13:20 and 13:25" (if this corresponds to the gap between Interval 1 and 2). Thus, this representation allows temporal constraints to be supported by general descriptions of the events, i.e., the person was mostly in the kitchen between 13:00 and 14:00, but not by more precise queries that we have reasons to doubt, e.g., being in the kitchen between 13:20 and 13:25.

3.3. *Constraints among multiple intervals*



(a) `START-START`$[l, u]$ (b) `PRECEDED-BY`$[l, u]$ (c) `OVERLAPS`$[l, u]$

Figure 4. Admissible intervals B of three quantitative Allen constraints given an interval i and a timespan $[l, u]$ that quantifies the algebra.

The representation introduced above would be useless unless it was also possible to propagate temporal constraints on the intervals defined by these sets. Fortunately this can be done, although under certain assumptions as we will see.

We can directly outline the admissible set of intervals B that a single interval i allows given a constraint as illustrated in Figure 4a. This figure shows one single interval i along with the set of intervals B that satisfies the temporal constraint i `Start-Start`$[l, u]b$, so that any interval b in B starts at least l and at most u time units after i (To make this example easy to understand we have used a disjunction of several Allen's interval constraints which we named `Start-Start`, it corresponds to (`Precedes` $\vee$ `Meets` $\vee$ `Overlaps` $\vee$ `Starts`)). Note that since this constraint does not limit the allowed end time of any interval in B, the set of allowed intervals stretches up towards infinity in the figure. For *mixed* constraints, i.e., constraints in which one interval's start time constrains another interval's end time or vice-versa, the geometric representation involves a projection onto the diagonal. An example of this

is illustrated in Figure 4b, the semantics here is that the end time of an interval i constrains the start time of another interval b. Thus, the start time of i is projected onto the diagonal to translate it into an end-time. The diagonal intersection is then used to constrain B in a similar way as in Figure 4a.

Furthermore, Figure 4c illustrates a constraint that involves more than a single time point in the STP, i.e., the start or the end time, taken from each of the intervals. Here, the start time of i limits the possible time of occurrence of the end time of B. The distinguishing factor here is that the start time of any b is also limited to occur after the start time of i.

3.4. *Propagation*

The propagation algorithm that is used to solve the context recognition problem is based on the the AC-3 algorithm [19], adapted to work on geometric sets of intervals. Like AC-3, the algorithm keeps a work list containing the arcs in the constraint network that should be propagated. This set is initialized to contain all the variables in the domain. Similarly, during propagation, arcs are removed from this list and processed. If this reduces the domain of a variable, arcs involving this variable are reintroduced into the list.

Algorithm 1

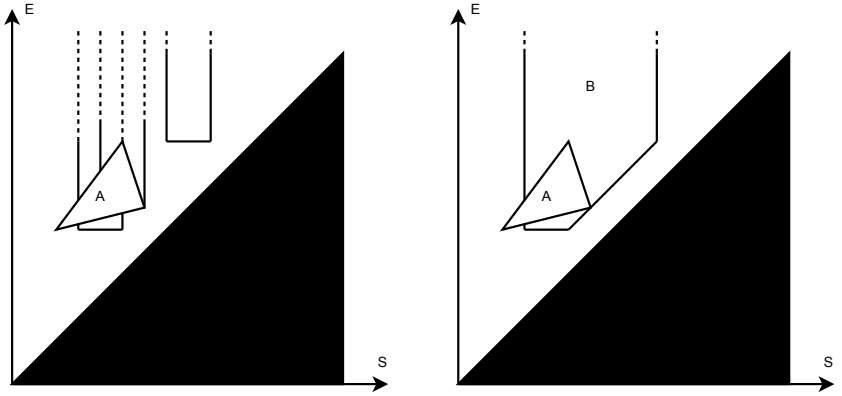
```
1: function propagate_arc(A, c, B)
2:     P ← ∅
3:     A^convex ← convex_subsets(A)
4:     for p in A^convex do
5:         I ← ∅
6:         for i in p do
7:             I ← I ∪ evaluate(i, c)
8:         end for
9:         p ← convex_hull(I)
10:        P ← P ∪ p
11:    end for
12:    B ← B ∩ P
13: end function
```

Algorithm 1 shows how we propagate one arc, $A \overset{c}{\leftarrow} B$, in the constraint network, i.e., how the algorithm removes values from the domain of a variable B with the help of the constraint c and the values in the domain of variable A. Traditionally, this is done by checking each value in the domain of B and searching for a variable in the domain of A that satisfies the constraint. If one exists, the value in B is kept, otherwise it is filtered. In our case, this is not possible since we do not maintain an explicit representations of the intervals in the domains of the variables. Instead,

given A and c, we calculate all possible values of B. This is done by calculating the *Minkowski Sum* [5] of the interval A and (dynamically) each convex set of intervals that are formed by evaluating the constraint on the vertex intervals in A. We will refer to this set as the *convolution* of A given c, and we will use it to geometrically intersect the previous domain of B in order to remove inconsistent values.

Algorithm 1 calculates $A \overset{c}{\leftarrow} B$ and takes as an argument two variables, A and B, and a constraint c. It starts by initializing a new empty set of polygons (line 2) that will be used to store the convolution of A. The next step is to calculate a convex decomposition A^{convex} of the polygons defining the set of intervals in A with the function *convex_subsets* (line 3). This function takes as input a set of possibly non-convex polygons and returns a larger (or equally sized) set of convex polygons that defines the same regions (line 4). This kind of decomposition is handled using an algorithm such as the one by Keil [17]. Note that optimal decomposition of a simple polygon can be done in $O\left(r^2n^2\right)$ time [17], where n is the total number of vertices and r is the number of notches (reflex angles). Note also that we can do this in $O\left(n\log\left(n\right)\right)$ time [15] with a guarantee that we do not get more than four times more convex pieces than the optimum.

The convolution itself is driven by solving a small STP containing two intervals (i.e., four time points). In the STP, these time points are initially constrained to each other with simple distance constraints reflecting the Allen interval constraint defined by c. Furthermore, one pair of time points are constrained to the start and end time of interval i respectively. The STP is then propagated with the Floyd-Warshall all pairs shortest path algorithm [10], which reduces the temporal domain of the remaining pair of time points.



(a) The individual convolutions of the intervals defined by region A with the constraints shown in Figure 4c.

(b) The convex hull of the individual convolutions shown in Figure 5a.

Figure 5. Convolution of a set of intervals.

At this stage, the mutual temporal relationship between the second pair of time

points in the STP reflects the set of intervals that are admissible given the constraint c and the interval i. This corresponds to the situation illustrated in Figures 4a–4c. In these figures, the pair of time points that are initially constrained represent the start and end time of i, and after propagation the remaining temporal flexibility in the second pair defines area B. The admissible region for B according to the interval i and the constraint c is then extracted by analyzing the remaining temporal flexibility of the second pair of time points. Each such convolution of a single interval i generates four new intervals (i.e., the vertices of the admissible area of B in Figure 4c). This process is done for each interval representing a vertex of A, and for each convex sub-polygon of A a set of intervals I is formed. This situation is shown in Figure 5a.

Redundant (interior) intervals are then removed from each I by taking the set's convex hull. This creates a (convex) polygon p (line 9) that defines the convolution of the convex subset of A, A^{convex}, with respect to the constraint c such as the one illustrated in Figure 5b. The convex hull is retrieved with a Graham scan [12] which has a complexity of $O(n\log(n))$ where n is the number of intervals (vertices) in I.

All such convex polygons form one set of polygons P that completely define the admissible values of B with respect to A and the constraint c. This set is then used to intersect the previous domain of B (line 12), which effectively removes values from B that cannot be satisfied with respect to A and the constraint c. Like the union, the intersection is calculated with a clipping algorithm such as the one outlined by Greiner and Hormann [13], which has a complexity of $O(mn)$, where m and n are the number of vertices (i.e., intervals) in the two polygons to be intersected. However, in practice, this can usually be done much faster if some sort of partitioning scheme is used.

It should be noted that since we are reasoning upon sets of intervals, not all constraints are of equal practical importance in AAIA; `MEETS` and `STARTS` are not very useful since they both imply equality between time points in an "off-by-one" fashion in the underlying STP. Making such distinctions is not very useful when reasoning upon sets of intervals. If, for instance, a set A contains all intervals such that $\forall a \in A : a_s \in [l, u]$ then the constraint B `MEETS` A constrains B so that $\forall b \in B : b_e \in [l, u]$ whereas B `OVERLAPS` A constrains B so that $\forall b \in B : b_e \in [l + \epsilon, u + \epsilon]$, where ϵ is the smallest representable unit of time. Given that we are reasoning on relatively large timespans, this distinction is not particularly useful to make in practice. (If a instead was one particular interval, then the resulting constraints at the STP level would be $b_e = a_s$ and $b_e \in [a_s, a_e] \bigwedge b_s < a_s$ respectively, where $i_s < i_e$ is implied for all intervals.)

Finally, just like in AC-3, when the domain of one variable is reduced, all of its outgoing arcs are reasserted in the work list. Furthermore, when the work list becomes empty the constraint network has been fully propagated. The algorithm outlined here provides the necessary functionality to propagate Allen's interval constraints in a network where the domains of the variables are sets of temporal intervals defined by polygons. The described algorithm performs the bulk work of the

CR within GiraffPlus, additional information about the same can be found in [34].

4. Query Language

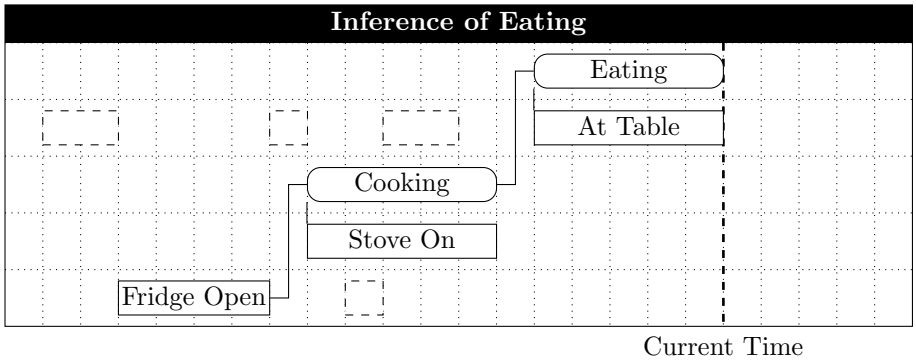


Figure 6. Example showing the dependencies used for inferring Eating. In this Figure, rectangles represents intervals containing sensor data while activities that are inferred by the system have rounded corners. In addition, a few dashed intervals have been added to highlight the fact that a timeline is a set of intervals.

GiraffPlus defines a query language that is used to post questions to the CR system. The language that governs how activities are inferred in GiraffPlus uses constraints in AAIA to describe how temporal intervals that represent activities are inferred from sets of intervals representing sensed data. A constraint used by the model might, for instance, declare that an activity, Eating, should be inferred whenever the inhabitant sits down AtTable shortly after Cooking. In this example AtTable can be inferred directly by preprocessing the sensor data coming from a pressure sensor on the chair, while Cooking, in turn, is an inferable activity which can be grounded in sensor data by requiring it to occur while the stove is on, and after the fridge has been opened. This example is illustrated in Figure 6.

This representation provides some flexibility in that "intermediate" activities such as Cooking can be reused or modified for different homes without rewriting the entire model. As can be seen in Figure 6 which illustrates the scenario mentioned above, a query for an activity is a tree like constraint network in which the leaves consists of data coming from sensors and where the intermediate nodes represents activities.

The CR service itself runs on a central server to facilitate upgrades and to allow for better control of access to the data. Context is recognized in response to queries made by the user through a client side API. A query consists of a rule document, an activity to be inferred, and a timespan of interest.

Listing 1 shows a sample rule document that can be used in the system. As can be seen, the format is based on XML. The inference of an activity is performed in three different steps, *preprocessing*, *inference* and *extraction* (illustrated in Figure 7), these

```
<?xml version="1.0" encoding="UTF-8" ?>
<rules home="testsite_se_2">

<preproc name="TunstallTrueFalse" in="Fridge" out="_fridge_open"/>
<preproc name="TunstallTrueFalse" in="Stove" out="_stove_on"/>
<preproc name="TunstallTrueFalse" in="Chair Kitchen" out="_at_table"/>

<rule out="_eating">
 <constraint from="_eating" type="after" to="_cooking" args="[0,600]"/>
 <constraint from="_eating" type="during" to="_at_table"/>
</rule>

<rule out="_cooking">
 <constraint from="_cooking" type="after" to="_fridge_open"
      args="[0,600]"/>
 <constraint from="_cooking" type="during" to="_stove_on"/>
</rule>

<extractor name="max" in="_eating" out="eating"/>
```

Listing 1. A rule document that describes how the activity eating can be inferred. When provided as an input to the CR system the the document creates the constraint network in Figure 6.

are represented in the document using the `preproc`, `rule` and `extractor` elements respectively. Each of these elements controls how the corresponding operations are performed on the CR server.

4.1. *The Context Recognition Service*

In GiraffPlus, context is recognized centrally by a REST servlet deployed on a Tomcat server. Queries to this service are done through a lightweight API which is embedded in several peripheral services that run on the client computer and 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.

Figure 7 shows the details of the context recognition service. The main point of interest is the inference procedure that is divided into three distinct steps; preprocessing, inference and extraction. The responsibilities of these are as follows:

Preprocessing module

On the server the client sends a query to the CR 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 timeline

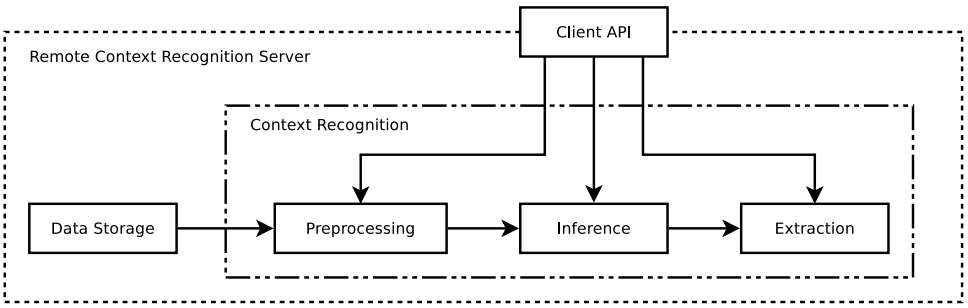


Figure 7. This figure shows the flow of data within the CR module.

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.

Listing 1 contains preprocessors that interpret boolean readings from sensors manufactured by the company Tunstall. This contrasts with the preprocessing of motion readings for instance, which only produces a boolean true reading but does not indicate the absence of movement. The approach is generic, however, since it does not impose any limitations on which sensor data can be used: symbolic or real-valued readings can be handled, and the preprocessors works on the entire set of samples contained in the timespan of the query simultaneously (i.e., not on individual samples). This module can, for instance, make use of the data coming from a temperature sensor placed on the warm-water pipe of the shower faucet to deduce that the inhabitant is showering. This requires a special logic which is implemented as a preprocessor since, after taking a shower the warm-water pipe remains warm for several hours.

Preprocessors are used to create the leaf nodes in the constraint network, which is later propagated by the *Inference module*. Line 4, for instance, creates a variable with the name "*_fridge_open*" in the document by utilizing data coming from the sensor named "Fridge" in home "testsite_se_2" (line 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 explained above, as temporal constraints in AAIA. Activities are inferred by performing temporal constraint propagation on the domains of intervals generated by the preprocessing module (arc consistency, see Section 3). The output is a domain of intervals that are admissible with respect to the rules.

The inferences step propagates the constraint network that is produced by the "rule" and "constraint" elements in the document. Line 8–11 in the Listing 1 for instance, creates the variable "eating" and uses temporal constraints to relate it to other variables (see Figure 6), either supplied by a preprocessor or by another rule. During inference, only the sensor data that is necessary to answer the query is fetched from the database. In addition, the CR system caches recently used data locally to facilitate (near) real-time viewing of activities as they unfold (which is accomplished by repeatedly polling the system).

Extraction Module

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. The extraction module's responsibility is to generate timelines that can be used by other software components (e.g., the visualization software or the alarm system).

In GiraffPlus this module only supports one type of extraction method, which extracts the maximum duration interval for an activity, an example of this can be seen on line 18 in Listing 1 which describes how the activity eating can be inferred from the variable "*_eating*". (This step can also be used to negate the timelines for instance.)

5. System Implementation

As stated, an important contribution of the GiraffPlus system is that it integrates components for environmental sensing, physiological sensing, context recognition, data visualization and storage. In this section we briefly present the components of the system and how they are integrated in order to provide a better understanding of the role and place of the CR in the overall architecture. Figure 8 shows how the components of the system system interacts with the services and hardware used in a typical deployment, the main components of which are the following:

5.1. *Physical environment*

The home of the end-user contains several sensors which are wirelessly connected to an Asus EEE Box PC which in turn is connected to Internet. The system is deployed in two steps, first a preliminary visit to the home is done together with caregivers to assess the needs of the inhabitant. During the visit the inhabitant is interviewed

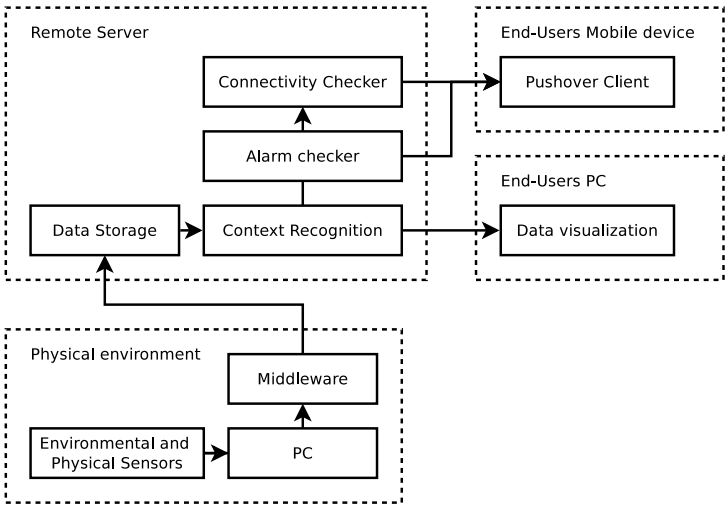


Figure 8. A high level overview of the relationship between the context recognition and the other components in the GiraffPlus system.

to get a firsthand account of his or hers daily activities. This information is then taken into account with the layout of the apartment and the system is installed on a subsequent visit a few days after the initial one. The vast majority of the work of configuring the system can be done off-site so that the installation procedure takes less than an hour in most cases.

Environmental and Physical sensors: The environmental sensors that are used include motion sensors, pressure sensors (to detect the presence of the inhabitant in the bed for instance), electrical usage sensors, reed-switch sensors (to detect open doors etc.), smoke alarms, flood detectors and more. All environmental sensors are provided by Tunstall, whereas the physiological sensors are provided by Intellicare. The latter measures physiological parameters such as blood pressure, heart rate and body weight. However, unless told otherwise, the end user takes these measurements whenever he or she so wishes, and the data these sensors provide are useful as-is to a caregiver and thus not deemed very interesting. Therefore, the focus is on using the environmental sensors (although there is no limitation on this, both types of data are handled equally).

PC: The PC is an Asus EEE Box PC, originally intended to be used as a media center, this computer is suitable to be used in homes due to its small form factor, low noise and power efficiency. The PC is connected to the Internet, either via a 3G router or directly depending on the available options at each test site. Furthermore, it is connected to a Tunstall Connect+ gateway which enables it to receive and forward data from the environmental sensors.

Middleware: The middleware handles forwarding of sensor data to the remote

database server and also buffering of data in case the Internet connection is temporarily lost or congested. Thus data that can not be submitted immediately is queued for later transmission. A detailed description of the middleware which is partially derived from the PERSONA project [32] can be found in [22].

5.2. *Remote Server*

The remote server hosts a database and several Java-servlets running on a Tomcat web server. This means that all connections to the server are done by Representational State Transfer (REST) HTTP calls. Since the data that is stored on the server is sensitive, all connections are protected with SSL and each user and home needs a personal certificate to connect to the server (i.e. a third party can not access the system using only a password).

Data Storage: The data storage provides an API to query and store information about homes in a MongoDB database. This includes the sensor samples that are sent from the homes and other data such as information about primary and secondary users and their access rights. MongoDB is a document database that focuses on scalability. Scalability was deemed an important feature since a useful and commercially viable system needs to be able to support thousands of homes.

Context Recognition: The CR system is deployed on the same server as the database in order to remove the overhead of transmitting raw samples over Internet and to facilitate updates. A client sends a query to the CR consisting of a rule document and a time period. The server in turn requests the required data from the DB and infers a corresponding timeline containing the activities that were queried for.

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

Connectivity Checker: This systems monitors the connectivity of the test sites and alerts technical support if a given home has not provided any data for a long period of time (due to issues with the Internet connection, the local PC or the sensor system).

5.3. *End user devices*

The end user, which can be an elderly a relative or a caregiver, can use the Giraff system with a PC or a smart mobile device.

Pushover Client: This software 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. In addition, the system supports acknowledging the message, although this functionality has not been fully implemented into the infrastructure yet.

Data Visualization: The PC in the home or at the caregivers office runs the data visualization and personalization software "DVPIS". It enables the user to fetch the elderly's physiological measurements (e.g. body weight blood pressure etc.) and perform activity queries to determine what the elderly has been doing during given periods of time.

6. Evaluation

6.1. *Benchmarks*

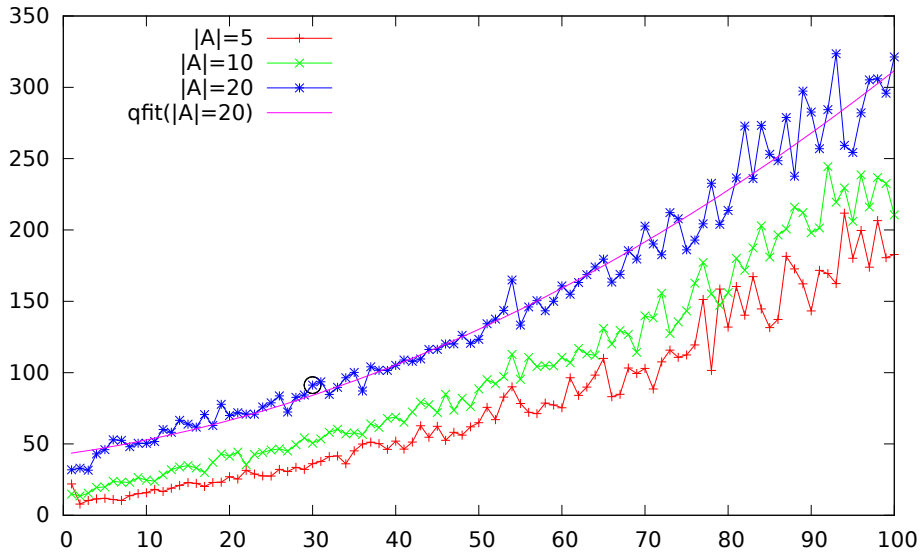


Figure 9. Time taken in milliseconds to fully "propagate" randomly generated networks consisting of two variables. The y-axis shows the average runtime of calculating $A \overset{c}{\leftarrow} B$ followed by $B \overset{c}{\leftarrow} A$, and in reverse order (i.e, AC-3's work order). The measurements are the average of 200 runs. The series denotes the number of sets in A and the x-axis denotes the number of sets in B. It took, for instance, roughly 90 ms on average to propagate the constraint network when variable A contained 20 sets of intervals and variable B 30 (i.e., problem instance $\langle |A| = 20, |B| = 30 \rangle$) and this is conveyed by the series and the x-axis respectively. The corresponding data point has been marked with a circle in the figure. Furthermore, a quadractic fit of series $|A| = 20$ has been included as "$qfit\,(|A| = 20)$".

In order to characterize the performance of the algorithm, the runtime was measured while solving a basic temporal constraint network with randomized variable

domains. Figure 9 shows the result of one such trial[a]. The figure shows how the complexity of solving a basic problem consisting of two variables, A and B, is affected by the increase in size of the domains of one of the variables. The three series each denote a different number of temporal sets in variable B, while the X-axis denote the number of temporal sets in variable A. Furthermore, each variable contains regions made up by several instances of a four vertex temporal set.

The experiment was conducted by repeatedly propagating $A \overset{c}{\leftarrow} B$ followed by $B \overset{c}{\leftarrow} A$ and subsequently $B \overset{c}{\leftarrow} A$ followed by $A \overset{c}{\leftarrow} B$ (as a separate problem), where c is the `CONTAINS` constraint (di). This was done 200 times for each point in the figure (100 in each order) and the timing is the average time taken. Thus, the runtime reflects the cost of making one arc in the constraint network consistent, effectively synchronizing the values of the two variables, which is the basic operation in the algorithm.

The fluctuations in the series can be explained by the fact that the algorithm is implemented in Java, and the garbage collector runs sporadically during the benchmarking. Also, since the domains are randomized their respective sizes does not describe the complexity of obtaining a solution completely since the random placements of the sets of intervals can be overlapping, thus creating slightly simpler problems, or more difficult (in case the overlap creates non-convex sets which needs to be decomposed by the algorithm before the propagation). Moreover, in each particular trial, e.g. $\langle |A| = 20, |B| = 30 \rangle$ the initial domains of A and B remains the same in all 200 trials, whereas $\langle |A| = 20, |B| = 31 \rangle$ for instance contains a completely new problem domain for *both* A and B.

The conclusion that can be drawn from this figure is that the complexity of solving a randomly generated problem is roughly $O(nm)$ as long as the problem is "sparse" (in the sense that the scale and number of intervals placed onto the timeline does not tend to generate overlaps). When the problem becomes more "dense", i.e. there exists a lot of overlap, then decomposition needs to be done more often and the complexity seems to grow as a quadratic polynomial, i.e., $O\left(n^2m^2\right)$. Set 20 in the data above is linear with an adjusted $R^2 = 0.9470$ and quadratic with an adjusted $R^2 = 0.9777$ and a quadratic coefficient of 0.0187 ($RMSE = 12.17$).

Solving equal sized problems on a sparser timeline would result in a lower quadratic coefficient and solving a more dense problem would result in a higher quadratic coefficient. Thus in the domain of CR, where the temporal size of the entire domain/timeline is closely related to the number of sets of intervals in it, the complexity of the algorithm is essentially linear with respect to the query window. I.e., inferring activities from two weeks of data takes toughly twice as much time as doing the same for one week (given that the data is placed uniformly on the timeline). This relationship is shown in Figure 10, as a basis for these series, 20 sets of intervals were consistently placed in domain A. The sets in the figure shows the

[a]Parts of the algorithm can be parallelized and solved by different threads concurrently, this functionality was however disabled during this experiment.

results of scaling the domain size by a factor of 0.01, 0.10, and 1.00 (i.e., reducing the size in the first two cases) while solving a problem with the same amount of randomly generated intervals. As can be seen, a lower domain size (i.e. 0.01), takes precedence over the other with respect to the runtime. Thus, 0.10 and 1.00 appears linear in comparison with the same sets of intervals placed randomly in the set with scale 0.01 which contains the most dense timelines.

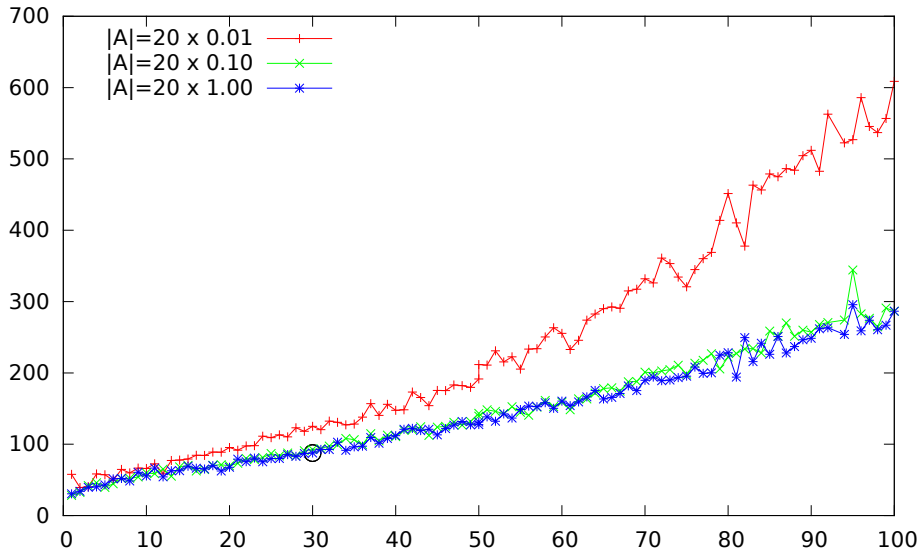


Figure 10. This figure shows the effect of scaling the temporal size of the entire domain while placing (consistently sized) randomly generated intervals on the timeline. The series with scale 1.00 corresponds to series $|A| = 20$ in Figure 9, although with a different initial pseudo-random seed.

6.2. *Test case in a Swedish Home*

Ground truth is difficult to obtain as it would require that the elderly themselves annotate their activities. An evaluation was therefore 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.

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. They also make his bed in the morning, and at lunchtime they heat his food in the micro oven (he

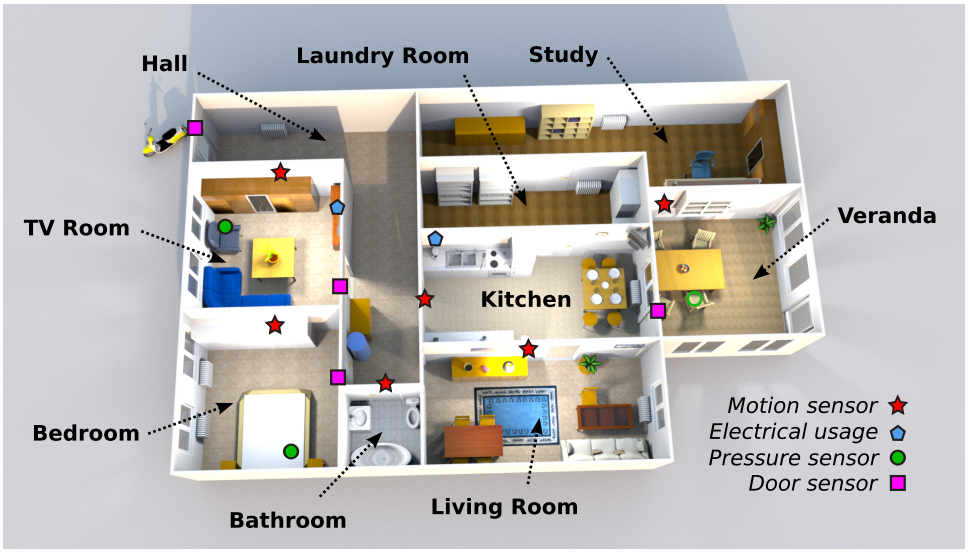


Figure 11. The layout of the second test site in Sweden. This is not an exact depiction but captures the general layout of the large home.

buys his food himself because he does not like the food they provide). The man's sons live nearby and visits him often, in addition, his grandchildren uses the Giraff telepresence robot to visit him remotely.

An initial working version of all software components of the system is available in the apartment. The apartment is depicted in Figure 11. 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 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[b]. 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[c], the output of this query is shown in Figure 12.

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

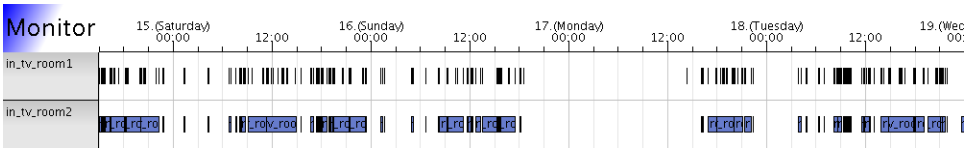


Figure 12. A pair of timelines showing when the elderly man visits the TV room, constructed using different methods of preprocessing the sensor data.

The topmost timeline, `in_tv_room1`, in Figure 12 shows the result of the first query. Given the fragmented nature of the timeline (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 timeline. In order to address this problem, another query was made using data from other motion sensors in the apartment as well, the output of this query is shown bottommost in Figure 12 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 timeline 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.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <rules home="testsite_se_2">
3
4  <preproc name="TunstallPIRSimple" in="PIR - TV Room" out="_in_tv_room1"
       args="" />
5  <preproc name="TunstallPIRSimple" in="PIR - TV Room, PIR - Bedroom, PIR -
       Kitchen" out="_in_tv_room2" args="" />
6
7  <extractor name="max" in="_in_tv_room1" out="in_tv_room1" />
8  <extractor name="max" in="_in_tv_room2" out="in_tv_room2" />
9
10 </rules>
```

Listing 2. A rule that infers when the person has been in the TV room using two different methods.

One responsibility of the CR module within GiraffPlus is to provide timelines 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

enough electricity in standby mode to be considered on.

shown in Listing 2.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rules xsi:noNamespaceSchemaLocation="rule_schema.xsd"
    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" />

<!-- ################## -->

<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>

<rule out="_awake_in_tv_room">
 <constraint from="_awake_in_tv_room" type="during" to="_in_tv_room2" />
 <constraint from="_awake_in_tv_room" type="after" args="[0,1000]"
     to="_in_bed" />
</rule>

<extractor name="max" in="_awake_in_kitchen" out="awake_in_kitchen" />
<extractor name="max" in="_awake_in_tv_room" out="awake_in_tv_room" />

</rules>
```

Listing 3. This listing shows an example rule that is used in the real system. The rule describes the conditions under which the activities *awake_in_kitchen* and *awake_in_tv_room* can be inferred.

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 timeline so

that each filtered interval occurred `AFTER` `in_bed` and `DURING` presence at the respective locations; `in_kitchen` and `in_tv_room`. The output of this query is shown in Figure 13.

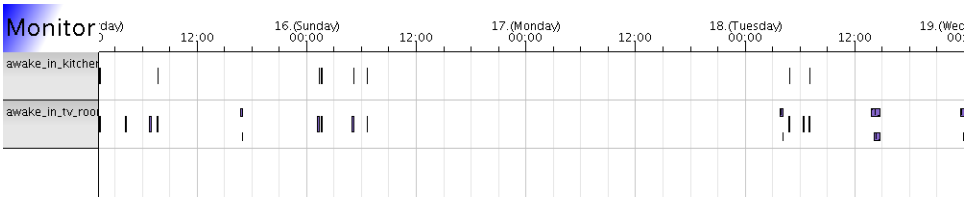


Figure 13. A pair of timelines showing when the elderly visits the kitchen and the TV-room after having left his bed.

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 Listing 3.

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.

7. Conclusions and Future Work

This paper has presented a fully working CR system that has been developed for the GiraffPlus project. The focus of the system is to address real world CR issues such as the scarcity of sensors and the need to be able to dynamically adapt the underlying inference model to the needs of the user, in this case the elderly and caregivers. The adaptability is achieved through the use of rule documents that control how sensory data is processed to an intermediate representation consisting of sets of intervals on timelines, how activities relate to this representation with the use of constraints from Allen's interval algebra, and how the inferred results are interpreted.

Surrounding tools in the system has also been discussed, such as the alarm system which provides caregivers and relatives with the ability to be notified when certain conditions occurs, and a monitoring system that sends notifications to system engineers in case of technical problems.

The system currently analyzes data coming from fifteen test sites (homes) located in three different countries. The complexity of the underlying algorithm has been evaluated in an experiment and was found to be linear in complexity for the problem encountered by the CR system. Furthermore, in two examples we have shown how

queries about the inhabitants behavior can be customized in cooperation with a medical professional. Future research will focus on expanding the capabilities of the constraint language and evaluating its usability in the different test sites. Furthermore, we will investigate the possibility of making the collected sensor data available to the research community.

References

1. Wolfgang Aigner and Silvia Miksch. Carevis: Integrated visualization of computerized protocols and temporal patient data. *Artificial Intelligence in Medicine*, 37(3):203–218, 2006. ISSN 0933-3657. doi: 10.1016/j.artmed.2006.04.002. URL `http://www.sciencedirect.com/science/article/pii/S0933365706000595`. Knowledge-Based Data Analysis in Medicine.
2. J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23 (2):123–154, 1984. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/0004-3702(84)90008-0.
3. J.C. Augusto and C.D. Nugent. The use of temporal reasoning and management of complex events in smart homes. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI)*, 2004.
4. Silvia Coradeschi, Amedeo Cesta, Gabriella Cortellessa, Luca Coraci, Javier Gonzalez, Lars Karlsson, Fransesco Furfari, Amy Loutfi, Andrea Orlandini, Filippo Palumbo, Federico Pecora, Stephen von Rump, Ales Stimec, Jonas Ullberg, and Britt Östlund. Giraffplus: Combining social interaction and long term monitoring for promoting independent living. In *6th International Conference on Human System Interactions (HSI)*, pages 578–585, 2013. doi: 10.1109/HSI.2013.6577883.
5. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry: algorithms and applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. ISBN 3-540-61270-X.
6. Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, May 1991. ISSN 0004-3702. doi: 10.1016/0004-3702(91)90006-6. URL `http://dl.acm.org/citation.cfm?id=120536.120554`.
7. 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.
8. G. Duftschmid, S. Miksch, and Gall. Verification of temporal scheduling constraints in clinical practice guidelines. *Art. Intelligence in Medicine*, 25(2):93–121, 2002.
9. T.V. Duong, H.H. Bui, D.Q. Phung, and S. Venkatesh. Activity recognition and abnormality detection with the switching hidden semi-markov model. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
10. Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5:345–348, June 1962. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/367766.368168. URL `http://doi.acm.org/10.1145/367766.368168`.
11. A. Goultiaeva and Y. Lespérance. Incremental plan recognition in an agent programming framework. In *Working Notes of the AAAI Workshop on Plan, Activity, and Intention Recognition (PAIR)*, 2007.

12. Ronald Lewis Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.
13. GÃ$\frac{1}{4}$nther Greiner and Kai Hormann. Efficient clipping of arbitrary polygons. *ACM Transactions on Graphics*, 17(2):71–83, April 1998.
14. R. Helaoui, M. Niepert, and H. Stuckenschmidt. Recognizing interleaved and concurrent activities: A statistical-relational approach. In *Proccedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2011.
15. Stefan Hertel and Kurt Mehlhorn. Fast triangulation of the plane with respect to simple polygons. *Information and Control*, 64(1-3):52–76, 1985.
16. V. Jakkula, D.J. Cook, and A.S. Crandall. Temporal pattern discovery for anomaly detection in a smart home. In *Proceedings of the 3rd IET Conference on Intelligent Environments (IE)*, 2007.
17. J. Mark Keil. Decomposing a polygon into simpler components. *SIAM J. Comput.*, 14(4):799–817, 1985.
18. L. Liao, D. Fox, and H. Kautz. Extracting places and activities from gps traces using hierarchical conditional random fields. *Robotics Research*, 26(1):119–134, 2007. ISSN 0278-3649.
19. Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977. ISSN 0004-3702. doi: 10.1016/0004-3702(77)90007-8. URL `http://www.sciencedirect.com/science/article/pii/0004370277900078`.
20. 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.
21. J. Modayil, T. Bai, and H. Kautz. Improving the recognition of interleaved activities. In *Proceedings of the 10th International Conference on Ubiquitous Computing (UbiComp)*, 2008.
22. Filippo Palumbo, Jonas Ullberg, Ales Štimec, Francesco Furfari, Lars Karlsson, and Silvia Coradeschi. Sensor network infrastructure for a home care monitoring system. *Sensors*, 14(3):3833–3860, 2014. ISSN 1424-8220. doi: 10.3390/s140303833. URL `http://www.mdpi.com/1424-8220/14/3/3833`.
23. D.J. Patterson, D. Fox, H. Kautz, and M. Philipose. Fine-grained activity recognition by aggregating abstract object usage. In *Proceedings of the 9th IEEE International Symposium on Wearable Computers*, 2005.
24. 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.
25. C. Pinhanez and A. Bobick. Fast constraint propagation on specialized allen networks and its application to action recognition and control. Technical Report 456, M.I.T. Media Lab, Perceptual Computing Section, 1998.
26. 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.
27. Daniele Riboni and Claudio Bettini. Context-aware activity recognition through a combination of ontological and statistical reasoning. In *Proceedings of the 6th International Conference on Ubiquitous Intelligence and Computing*, UIC '09, pages 39–53, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-02829-8. doi: 10.1007/978-3-642-02830-4_5.
28. Jean-Francois Rit. Propagating temporal constraints for scheduling. In *Proceedings of the 5th National Conference on Artificial Intelligence*, 1986.
29. Geetika Singla, Diane J. Cook, and Maureen Schmitter-Edgecombe. Recognizing independent and joint activities among multiple residents in smart environments. *Ambient Intelligence and Humanized Computing*, 1(1):57–63, 2010.
30. T. Springer and A.-Y. Turhan. Employing description logics in ambient intelligence for modeling and reasoning about complex situations. *Ambient Intelligence and Smart Environments*, 1(3):235–259, 2009.
31. Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120:248–253, 1998.
32. Mohammad-Reza Tazari, Francesco Furfari, Juan-PabloLÃ¡zaro Ramos, and Erina Ferro. The persona service platform for aal spaces. In Hideyuki Nakashima, Hamid Aghajan, and JuanCarlos Augusto, editors, *Handbook of Ambient Intelligence and Smart Environments*, pages 1171–1199. Springer US, 2010. ISBN 978-0-387-93807-3. doi: 10.1007/978-0-387-93808-0_43. URL `http://dx.doi.org/10.1007/978-0-387-93808-0_43`.
33. E. Tsang. *Foundations of constraint satisfaction.* Computation in cognitive science. Academic Press, 1993. ISBN 9780127016108. URL `http://books.google.se/books?id=TnxQAAAAMAAJ`.
34. Jonas Ullberg, , and Federico Pecora. Propagating constraints on sets of intervals. In *ICAPS Workshop on Planning and Scheduling with Timelines (PSTL)*, 2012.
35. J. Wu, A. Osuntogun, T. Choudhury, M. Philipose, and J. Rehg. A scalable approach to activity recognition based on object use. In *Proceedings of ICCV 2007*, 2007.

Appendix B

Noname manuscript No.
(will be inserted by the editor)

Towards Configuration Planning with partially ordered preferences: representation and results

Lia Susana d.C. Silva-Lopez* · **Mathias Broxvall** · **Amy Loutfi** · **Lars Karlsson**

Received: date / Accepted: date

Abstract Configuration planning for a distributed robotic system is the problem of how to configure the system over time in order to achieve some causal and/or information goals. Such a configuration plan specifies what components (sensor, actuator and computational devices), should be active at different times and how they should exchange information.

In this paper we present theoretical and preliminary empirical results of an approach to configuration planning that incorporates general partially ordered preferences in order to distinguish preferred configuration plans from less preferred ones. As there can be multiple preference categories, the planner solves a multiple-objective optimization problem: for a given problem, it finds all possible valid, non-dominated configuration plans.

The system has been able to successfully cope with partial ordering relations between quantitative preferences in practically acceptable times, as shown in the empirical results. Preferences here are represented as c-semirings, and are used for establishing dominance of a solution over another in order to obtain a set of configurations that will constitute the solution of a configuration planning problem with partially ordered preferences (C3PR). The dominance operators tested in this paper are Pareto and Lorenz dominance. Our solver considers one guiding heuristic for obtaining the first solution, and then switches to a dominance based monotonically decreasing heuristic used for pruning dominated partial configuration plans. In our preliminary empirical results, we perform a statistical study in the space of problem instances and establish families of problems for which our approach is computationally feasible.

Center for Applied Autonomous Sensor Systems, Örebro University, SE-70182 Sweden
E-mail: lia.silva@gmail.com*
{mathias.broxvall,amy.loutfi,lars.karlsson}@oru.se

1 Introduction

Consider a distributed robotic systems consisting of a set of communicating components capable of sensing, actuation and/or information processing. These may involve various types of sensors, from simple ones like temperature sensors, or pressure sensors in furniture, to advanced sensing and information processing systems like computer vision systems. They may also involve various types of actuators, like automatic door openers and radiators, and complex robots with varying degrees of autonomy and capable of both sensing (i.e. cameras) and actuation (i.e mobility) as well as information processing (i.e. localization, path planning). Finally, the different components can exchange information by means of a network, which may involve both wired and wireless communication.

A snapshot of all running sensing, actuation, information processing and communication tasks given to the different subsystems in order to solve some overall task is then called a configuration of the system. Different tasks may require different configurations, and even for the same task there may be several different configuration plans that could solve it. Some tasks may also require several steps to solve, this means that different components will be active at different times. In other words, the configuration can change over time. In this context, the configuration planning problem is that of finding a configuration plan that specifies the configuration of the system over time and that will take the system and the environment from an initial state to a state satisfying some predetermined criteria – given an accurate domain description of the components, the tasks they can perform and their effect on the environment. Note that the target states, that is the goal, may involve that certain information is produced (e.g. about the position of a person) in addition to the physical state of the environment (e.g. that a robot is at a certain position).

Configuration planning can be seen as an extension of task planning [15]. A task plan essentially is a sequence of actions where each action from the planner's perspective is considered an indivisible unit, such as *pickup box1* or *move robot from room1 to room3*. The different actions in a task plan are causally dependent on each other. For instance, if *box1* is in *room3* and the *robot* initially is in *room1*, then the action *pickup box1* needs to be preceded by the action *move robot from room1 to room3*. This is expressed in the planning domain as a precondition of *pickup box1* and an effect of *move robot from room1 to room3* and can be of the form *in robot room3*. An admissible plan is one where each action's preconditions are satisfied in the state in which the action is to be performed. A task planning problem consists of an initial state, a set of goals and a set of available actions, and a solution is an admissible plan that leads to a state where the goals are satisfied.

Configuration planning can in addition to causal interactions also involve parallel execution of tasks on components and information flows between tasks and components. In an admissible configuration plan, both causal and information requirements of each component in each step must be satisfied. Thus, an action in task planning corresponds in configuration planning to a temporary configuration. Actually, a previous approach to configuration planning by Lundh et al [18] utilized a task planner to generate a skeleton plan which satisfied causal requirements, and then a configuration planner transformed each action into a configuration. The end result was a plan consisting of a sequence of configurations. However, the approach presented in this paper integrates reasoning about causal and information requirements into a single algorithm.

The work presented in this article was done in the context of the GiraffPlus system [12], for support of old people at home, where the system, consisting of a range of sensors (e.g. motion, light, door closing, temperature, pressure in/under furniture) and a semi-autonomous robotic platform, needs to be configured in order to monitor specific everyday activities. However, the same techniques should also be applicable to many other domains such as monitoring and aiding people at department stores or airports, or managing and supervising warehouses. We strive for an approach with general applicability.

Furthermore, we often seek plans which are not only admissible, but also optimal with respect to some given preference function which specify to what degree a solution is preferred considering a number of criteria (preference categories). This preference function may take into account both conditions internal to the system, such as the appropriateness of different actions/components for achieving some effect, or externally imposed conditions like a user's ranking of soft goals. For most quantitative languages for preferences, the ordering relations between preferences are total. However, partial orders for preferences are desirable as many real-life configuration problems often involve incomparable preference criteria as well as cases in which no dominance between criteria exists.

A configuration planning with partially-ordered preferences (C3PR) problem is that of finding the set of non-dominated admissible configurations given a partially-ordered preference function over the set of configurations and a domination criteria. A solution to a C3PR problem is the maximum set of fully admissible configurations that are non-dominant between each other according to the goal preferences, but more dominant than any other fully admissible configuration.

To illustrate the need of partially ordered preferences, consider a request for the shortest and least expensive configuration. Suppose the planner returns three dominant (see more on dominance functions in the section of dominance functions) configurations that are non-dominant between each other: one with cost 5 and length 5, one with cost 4 and length 6, and one with cost 6 and length 4. If only one configuration is selected from the set of configurations in the solution, and we were to apply a bias function to that favored a configuration that has a more balanced set of scores, then the first configuration would be ranked as the best one. If on the other hand the preferences were given using the total order of either of the two criteria, we would have had ended up either with the second or the third configuration, depending on which criteria we decided to favor. Partial orders are not only desirable to represent features of an application domain in which a representation with total orders becomes artificial and even forced, but they are also flexible enough for allowing the use of total orders over the non-dominant solutions in case a ranking that considers other criteria is needed.

The contributions of this paper are three. First, the C3PR problem is explained and a representation for it is presented (Section 3). Second, an algorithm, heuristics and dominance operators for solving C3PR problems (Section 4). Third, empirical tests over an implementation of the second and third contributions of this paper (Sections 5). The authors have made both the implementation and the data available to the public. Conclusions are in Section 6.

2 Related Work

Regarding configuration planning, we adopt the problem definition given by Lundh et al. [18]. Historically, the problem described by Lundh is within the context of achieving unattended and adaptable collections of sensors, actuators and programs with cognition and communication capabilities, in which the notion of robot *emerges* from the interaction between the elements of the collection. ASyMTRe [20] is a related approach for configuring coalitions of single-task robots performing multi robot tasks using instantaneous

assignment. According to the taxonomy by Korsah et al. in [17], the configuration planning problem is more related to the "multi-task robots, multi-robot tasks" category, while considering that there are many different ways in which the utility can be calculated, which in turn may end up in different degrees of interdependence of agent-task utilities.

Regarding planning with preferences, Baier and McIlraith [3] made a comprehensive survey and in the following we will use their distinctions. Quantitative preference formalisms associate plans with numerical preference values. In Markov Decision Processes [23] for planning in stochastic domains, action-state combinations come with different rewards which are aggregated over time, often taking a discount into account. Policies, that is plans that are conditional on the state, that have maximal expected discounted rewards are preferred. These policies are typically generated by using dynamic programming methods.

PDDL3 [14] is a formalism that also supports quantitative preferences. Preferences in PDDL3 can refer to conditions at specific points or periods of time, or to temporal relations between conditions. For each preference, it is determined whether it is violated, and a metric function computes an aggregated numeric value for the set of preferences, which serves as the preference value for the entire plan. The forward-chaining heuristic planner HPLAN-P [2] is a notable planner for PDDL3.

Qualitative preference formalisms are typically based on specifying a preference ordering on choices. In conditional preference networks [7], each node represents a choice (e.g. what to have for desert) and how the preferences for that choice depend on other choices (e.g. what to have for main course). There are also extensions to those such as Tradeoff-enhanced CP-nets [10], which allows to express the relative importance of choices (e.g. the choice of main course is more important than the choice of beverage when dining at a highly rated restaurant). For the latter, the planner PREFPLAN [8] has been developed. Finally, there are qualitative formalisms for temporally extended preferences, such as PP [25] and *LPP* [4] which utilizes linear temporal logic [22]. PPLAN [5] is one example of a planner for LPP.

In contrast to the quantitative formalisms reviewed here, the formalism proposed in our paper allows planning for *multiple* partially ordered quantitative preferences. We got inspiration from Brafman et al, who stated in [9] that the application to planning with preferences of the work of Bistarelli et al in [6] has not been discussed, and pointed its value as a future research topic. To the best of out knowledge, ours is the first work that integrates concepts from Bistarelli's framework into planning with preferences.

3 Representing problem instances

Before we can come to the representation and definition of the computational problem of configuration planning with preferences we will give an informal definition of the problem. This definition is given in the context of a networked robotic environment that consists of a number of *functionalities* where each functionality can sense/act on the environment or perform pure computations.

Consider a scenario in which we have a robot that is too simplistic to have any localization sensors on it, however it does have two functionalities: a navigation functionality that can move iff it is given localization data, and an onboard temperature sensor. Additionally we have a number of sensors distributed in an elderly care home that are capable of localizing robots and humans alike.

In order for the robot to navigate the home, we need a plan that involves simultaneous actions in both the robot itself and in the sensors which need to track the robot and send the localization information to it in order to close the control loop. Thus we need to represent not only *casual* state variables that represents the state of the environment, but also *information* state variables that represent continuous information exchange between entities. With this representation we can have goals that are of a casual nature (eg. the robot should be in a location) or of an information nature (eg. send temperature measurements from a location).

An example would be a plan that uses the navigation component of the robot, the localization systems, and finally the onboard temperature sensor on the robot to take a temperature measurement in the bedroom.

1. Run navigation on robot to move from kitchen to hallway, requires localization data. Run tracking sensor in kitchen to generate localization data, send to navigator.
2. Run navigation on robot to move from hallway to bedroom, requires localization data. Run tracking sensor in hallway to generate localization data, send to navigator.
3. Run temperature sensor on robot, gives temperature in bedroom. Output as requested goal.

To represent the state of the environment we will use *state variable assignments* to represent all casual state variables (eg. `location(robot) = bedroom`) and state variables without assignments to represent information that is being sensed (eg. `localization(robot)`).

We can then represent functionalities as having a number of *requirements* and acting as *sources* of either information state variables and/or of casual state variables.

The temperature sensor from above can be described as having a requirement `location(robot) = x` and the effect of being a source of `temperature(x)` data – where `x` can be any room of the environment.

Finally, different functionalities can come with costs (or utilities) in a range of different metrics such as power con-

sumption, noise level, reliability that depend on the given (required) information or casual state variable (assignments). We call these metrics *preference categories* and build a partial-ordered final preference value from them. This allows for the planner to output only plans that are optimal depending on a selected definition of optimality.

In the remaining of this paper, whenever an element o needs to be extracted from a tuple t that contains such a named entry we will write $\pi_o(t)$ or simply o if t is implicit.

3.1 Formal representation of C3PR problems

We give here a mathematical definition of what a configuration plan is, define functionalities, admissibility of configuration plans and methods for expressing and computing preferences over a domain $\mathbf{U}_{pref}$. Finally we define the computational problem of finding the optimal admissible configuration plans.

Definition 1 A C3PR problem instance is a tuple:

$$C3PR = \langle \mathbb{F}, \mathbb{D}_N, \mathbb{D}_A, \mathbf{U}_{pref}, g \rangle$$

In which $\mathbb{F}$ is a set of functionalities, $\mathbb{D}_N$ is a set of all possible state variables names and $\mathbb{D}_A$ is a set of all possible state variable assignments, $\mathbf{U}_{pref}$ is the cross product of the set of possible values of all preference categories, and g is a goal.

The state variable assignments must each contain a state variable name and a value: $\mathbb{D}_A \subseteq \mathbb{D}_N \times D$ for some set of values D. A goal is a set containing one or more state variable names or state variable assignments.

State variable names are used to represent observable (information) or controllable (casual) properties, e.g. the temperature in a room or the location of the robot:

$$\mathbb{D}_N = \{\texttt{temp(bedroom)}, \texttt{loc(rob)}, ...\}$$

The state variable assignments are the set of all possible values for the state variables that can be controlled:

$$\mathbb{D}_A = \{\texttt{loc(rob)} = \texttt{bed}, \texttt{loc(rob)} = \texttt{kitch}, ...\}$$

Note that $\mathbb{D}_N$ and/or $\mathbb{D}_A$ can be infinite of size since we separate the definition of the problem from the language in which the problem is expressed. Practically, a simple STRIPS [13] inspired language with classical variables and unification is used for expressing a C3PR problem.

Definition 2 A functionality $F \in \mathbb{F}$ is a tuple:

$$F = \langle \mathbf{I}_r, \mathbf{C}_r, \mathbf{I}_s, \mathbf{C}_s, \mathbf{D}, \mathbf{P}_r, \mathbf{P}_s \rangle$$

In which $\mathbf{I}_r$ and $\mathbf{C}_r$ are sets of unique functionality variables representing ungrounded information and causal requirements, and $\mathbf{I}_s$ and $\mathbf{C}_s$ are sets of unique functionality variables representing ungrounded information and causal sources. $\mathbf{D}$ is a set of mappings from functionality variables to state variables or state variable assignments. This set constrains the mappings allowed by the functionality.

$$\mathbf{D} \subseteq (\mathbb{D}_N \cup \mathbb{D}_A)^{\mathbf{I}_r \cup \mathbf{C}_r \cup \mathbf{I}_s \cup \mathbf{C}_s}$$

We define two *preference functions*, one for the requirements $\mathbf{P}_r$ and one for the sources $\mathbf{P}_s$:

$$\mathbf{P}_r : \mathbf{I}_r \cup \mathbf{C}_r \rightarrow \mathbf{U}_{pref}$$

$$\mathbf{P}_s : \mathbf{I}_s \cup \mathbf{C}_s \rightarrow \mathbf{U}_{pref}$$

The significance and use of the preference functions are detailed in Section 3.2.

Finally, we define a *functionality instance* as a copy of a functionality with all variables substituted for a new unique variable name and all constraints updated accordingly.

To continue the earlier example, we can describe a functionality such as the navigation functionality that can navigate from room `R1` to room `R2` iff it receives localization information for room `R1`. For brevity we limit the choices of `R1` to `bed` or `hall` and of `R2` to `bed` only.

$$\mathbf{I}_r = \{x\} \quad \mathbf{C}_r = \{y\} \quad \mathbf{C}_s = \{z\}$$

$$\mathbf{D} = \left\{ \begin{Bmatrix} x \mapsto \texttt{track(rob,hall)}, \\ y \mapsto \texttt{loc(rob)} = \texttt{hall}, \\ z \mapsto \texttt{loc(rob)} = \texttt{bed} \end{Bmatrix}, \begin{Bmatrix} x \mapsto \texttt{track(rob,bed)}, \\ y \mapsto \texttt{loc(rob)} = \texttt{bed}, \\ z \mapsto \texttt{loc(rob)} = \texttt{bed} \end{Bmatrix} \right\}$$

Thus the set of functions $\mathbf{D}$ allows for parameterization of the functionality (can start in any of the two rooms) and unification (starting room dictates which tracking data is needed). Again, keep in mind that $\mathbf{D}$ is given more compactly by the syntax of the language used for describing functionalities.

It is here important to not confuse the functionality variables (x,y,z) that only describes which requirement/source of a functionality we are talking about with the state variables (`loc(rob)=bed`) which describe the required/provided state information or state of the environment.

Since we require configuration plans to contain functionality instances that can be executed multiple times at different timepoints, we require a way to represent their execution timepoint(s) and the (potentially) varying parameters and assignments to state variables. For this we require each functionality instance to contain a unique set of variables by renaming from an existing functionality. Furthermore, we require a set of execution ordering constraints over these instances. This is expressed in a temporal reasoning algebra $\mathscr{R}$ such as the point algebra **PA** [26], Simple Temporal Problem **STP** [11] or Allens intervall algebra **IA** [1]. For the remainder of this paper, unless stated otherwise we will use the point algebra as $\mathscr{R}$.

Definition 3 A configuration plan C is a tuple:

$$C = \langle \mathbf{Fi}, Lk_i, Lk_c, \mathbf{Ex} \rangle$$

Where **Fi** is a set of functionality instances used in this plan. Lk_i and Lk_c are linking functions that given an information or a causal requirement gives the variable name of a corresponding source in some other functionality instance.

Ex is a set of execution order constraints over **Fi** so that $\mathbf{Ex} \subseteq \mathbf{Fi} \times \mathbf{Fi} \times \mathscr{R}$, in which $\mathscr{R}$ is a set of temporal relations used when describing the execution order constraints. Consistency of **Ex** is determined by global consistency for the chosen relational algebra $\mathscr{R}$.

Definition 4 Given a configuration plan C, let $\mathbf{Fi} = \pi_{\mathbf{Fi}}(C)$ and let $\mathbb{I}_r$, $\mathbb{C}_r$,$\mathbb{I}_s$, $\mathbb{C}_s$ be the union of all information variables and causual requirement/source variables in C:

$$\mathbb{I}_r = \bigcup_{Fi \in \mathbf{Fi}} \pi_{\mathbf{I}_r}(Fi) \quad \mathbb{C}_r = \bigcup_{Fi \in \mathbf{Fi}} \pi_{\mathbf{C}_r}(Fi)$$

$$\mathbb{I}_s = \bigcup_{Fi \in \mathbf{Fi}} \pi_{\mathbf{I}_s}(Fi) \quad \mathbb{C}_s = \bigcup_{Fi \in \mathbf{Fi}} \pi_{\mathbf{C}_s}(Fi)$$

A *linking function* for C is a total function that given any requirement belonging to a $Fi \in \mathbf{Fi}$ returns the source that has been assigned to satisfy it.

$$Lk_i : \mathbb{I}_r \to \mathbb{I}_s \quad Lk_c : \mathbb{C}_r \to \mathbb{C}_s$$

We say that a pair of linking functions Lk_i, Lk_c are *satisfiable* if and only if there exist two mappings $M_i : \mathbb{I}_r \to \mathbb{D}_N$, $M_c : \mathbb{C}_r \to \mathbb{D}_A$ such that:

$$M_i(x) = M_i(Lk_i(x)) \quad M_c(x) = M_c(Lk_c(x))$$

$$\forall Fi \in \mathbf{Fi} : \exists d \in \pi_D(Fi) : \forall x \in \mathbf{I}_r \cup \mathbf{I}_s : d(x) = M_i(x)$$

$$\forall Fi \in \mathbf{Fi} : \exists d \in \pi_D(Fi) : \forall x \in \mathbf{C}_r \cup \mathbf{C}_s : d(x) = M_c(x)$$

We note that the definition of link satisfiability requires two parts: firstly that the mapping of state variable name or state variable assignment to the requirements must match the corresponding mapping to the linked source; secondly, that the allowed assignments to requirements and sources are restricted by the domain **D** given by the corresponding functionalities.

To continue the example, assume that we have a second functionality `kinnect`:

$$I_r = \{\} \quad C_r = \{\} \quad I_s = \{w\}$$

$$D = \{\{w \mapsto \texttt{track}(\texttt{rob}, \texttt{hall})\}\}$$

We can then create a linking function $Lk_i : x \mapsto w$. Assuming that we have a starting condition given by a functionality:

$$I_r = \{\} \quad C_r = \{\} \quad I_s = \quad C_s = \{a\}$$

$$D = \{\{a \mapsto \texttt{loc}(\texttt{rob}) = \texttt{hall}\}\}$$

We can create linking function $Lk_c : y \mapsto a$. These two linking functions together are satisfiable since there exist mappings:

$$M_i : x, w \mapsto \texttt{track}(\texttt{rob}, \texttt{hall})$$

$$M_c : y, a \mapsto \texttt{loc}(\texttt{rob}) = \texttt{hall} \;\; z \mapsto \texttt{loc}(\texttt{rob}) = \texttt{bed}$$

3.2 Representing preferences

As described in Section 1, many configuration problems can be described by a multitude of incomparable optimization criteria – something which is expressed as one or more properties of the functionalities. For example, a functionality may take 5 seconds, consume 1 unit of a resource and have a reliability of 90%.

In the interest of avoiding context dependent weighting functions we take the approach of partial orderings from the multivariable optimisation community. For this purpose we introduce the following three concepts for configuration planning: (i) a *preference category* representing one type of information eg. cost or reliability; (ii) a *preference* value consisting of an algebraic value for each such category, eg. 50$ and 90% success; and finally (iii) a *preference relation* that compares preferences. An example of the later would be the expression "50$ at 20% chance is better than 5$ at 10% chance".

For the first, preference categories, we will consider a generalization that is capable of representing a wide range of different types of information. We define each preference category as consisting of values from a corresponding c-semiring with together with a comparison operation. As such this lends itself well to expressing everything from crisp costs, to fuzzy logic, to probabilities.

We recall that a semiring is an algebraic structure containing a set **A**, and two binary operations corresponding to addition and multiplication [6]:

$$X = \langle \mathbf{A}, +, \times \rangle$$

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 ($\forall a \in \mathbf{A} : a + a = a$), a commutative $\times$ operation, and 1 as the absorbing element in $+$, then such semiring is a c-semiring [6]. To emphasize the idempotency we will sometimes write $\bot$ for the additive unity 0 and/or $\top$ for the multiplicative unity 1. We write a c-semiring S as:

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

For the preference function in the C3PR planner we require the domain to specify a corresponding semiring and operations on them for each preference category.

Definition 5 A preference category represents a type of preference that can exist in a C3PR problem and is given by a tuple $< S, \prec_S >$ where S is a c-semiring and $\prec_S$ is a (partial) order over the elements of S.

The preferences for a given planning problem is a tuple of such preference categories $< C_1, \ldots, C_n >$ corresponding to a semiring $\mathbf{U}_{pref}$:

$$\mathbf{U}_{pref} = \langle \mathbf{A}_1 \times \ldots \times \mathbf{A}_n, +, \times, 0, 1 \rangle$$

where $A_1, \ldots, A_n$ are the values of the corresponding semirings $C_1, \ldots, C_n$ and where the operators and additive / multiplicative unity is componentwise defined by the semirings.

We define a *partial* ordering $\prec$ over the preferences as:

$$x \prec y \Leftrightarrow \pi_1(x) \prec_{S_1} \pi_1(y) \wedge \cdots \wedge \pi_n(x) \prec_{S_n} \pi_n(y)$$

Definition 6 For a given configuration plan C, we say that the preference requirements are satisfied iff for each functionality Fi holds:

$$\forall x \in I_r : P_r(x) \prec P_s(Lk_i(x))$$

$$\forall x \in C_r : P_r(x) \prec P_s(Lk_c(x))$$

Where P_r and P_s are the preference functions for functionality Fi respectively the functionality linked to by $Lk(x)$.

Note that the notion of preference satisfaction is only used to further restrict the possible sources for a requirement. In order to compute preference values for complete configuration plans we need:

Definition 7 For a given configuration plan C, the preference value $\mathbf{pref}(C)$ is given by:

$$\mathbf{pref}(C) = \prod_{x \in I_r} P_s(Lk_i(x)) \times \prod_{x \in C_r} P_s(Lk_c(x))$$

We note that the domain of $\mathbf{pref}(C)$ is $\mathbf{U}_{pref}$ with the partial order $\prec$ used to differentiate between admissible configurations. Furthermore, the preference value of a configuration plan is monotonically decreasing as additional functionalities and links are added to it.

To extend the example with preferences, assume that $\mathbf{U}_{pref}$ is the cross product of two c-semirings named *cost* and *reliability*, both with a domain $[0,1]$ and max respectively min as addition and multiplication. We can then encode that the navigation functionality requires at least 0.7 reliability on localization data and ignores cost:

$$P_r : x \mapsto \langle \bot, 0.7 \rangle \quad y \mapsto \langle \bot, \bot \rangle$$

Similarly we can encode a cost and reliability of 0.5 for the functionality effects:

$$\texttt{navigator} : P_s : z \mapsto \langle 0.5, 0.8 \rangle$$

$$\texttt{kinnect} : P_s : w \mapsto \langle \top, 0.7 \rangle$$

The same linking functionalities as above are still satisfiable, and the preference value for the final configuration plan is $\langle 0.5, 0.7 \rangle$

Definition 8 A fully admissible configuration plan is a configuration plan C_a in which the linking function Lk is satisfiable and total, C_a is preference satisfied, **Ex** is consistent, and for at least one linearisation, the following conditions for a discrete, totally ordered timeline hold:

- For every pair of functionalities fi_1, fi_2: if fi_2 is linked by Lk_i to fi_1 then fi_1 is executed during the execution of fi_2. For **PA** we require the equality relation, for **IA** we require any of the non-strict during relations.
- For every pair of functionalities fi_1, fi_2: if fi_2 is linked by Lk_c to fi_1 for some causual requirement $x \in C_r(fi_1)$ then fi_2 is executed before the execution of fi_2 and there exist no other functionality f_3 executing in between and with a causual source effecting the same state variable assignment.

When a configuration plan C_a does not fulfill all of the former conditions, then such a configuration is a *partial configuration plan*.

As we can see from the above definition we now have a way of defining a C3PR problem instance, defining what is an admissible configuration plan (solution) and evaluating a *partially ordered preference* value for each such configuration plan.

3.3 The C3PR optimization problem

For many C3PR problem instances there exists a large potential number of fully admissible configuration plans. In order to minimize the number of choices returned from a solver for the C3PR problems we require an *optimality* criteria. One of the most common optimality criteria for partial orders is *Pareto optimality* which is derived directly from the partially ordered preference function above. We note that the computational problem of interest here is thus a multi-objective optimization problem, giving a set of mutually independent optimal solutions. Since there exists further definitions of optimality criteria we define the optimization problem with respect to a general optimality criterion:

Definition 9 The C3PR optimization problem is defined as follows: given a C3PR problem instance $C3PR$ and a dominance operator $\mathscr{R}_d$ calculate the *maximum* set G of fully admissible configuration plans such that no configuration plan in G is dominated through $\mathscr{R}_d$ by any other element of G.

$$G = C3RP_d(C3PR, \mathscr{R}_d)$$

$$c_1, c_2 \in G \rightarrow \neg(c_1 \mathscr{R}_d c_2)$$

We note that when $\mathscr{R}_d$ is the empty relation, the solution G is the full set of all admissible configuration plans for the given C3PR problem instance.

As we can see, a dominance relation $\mathscr{R}_d$ in a C3PR problem is a preference relation for establishing if a configuration plan C_x is more preferable or just as preferable as a configuration plan $C_{x'}$.

Definition 10 C3PR$_{\mathscr{P}}$ (Pareto) Problem

A C3PR Pareto problem C3PR$_{\mathscr{P}}$ is a C3PR optimization problem in which the dominance relation $\mathscr{R}_d$ is a Pareto dominance relation. $\mathscr{R}_d$ can be weak Pareto dominance $\succeq_{\mathscr{P}}$ or strict Pareto dominance $\succ_{\mathscr{P}}$.

Assume that the domain of preferences is:

$$\mathbf{U}_{pref} = \langle \mathbf{A}_1 \times \ldots \times \mathbf{A}_n, +, \times, 0, 1 \rangle$$

Then, weak Pareto dominance $c_1 \succeq_{\mathscr{P}} c_2$ occurs if and only if each preference category have a non-strictly higher value in c_1 than c_2 [16, 19]:

$$\forall A_i : \pi_{A_i}(c_1) \geq \pi_{A_i}(c_2)$$

Strict Pareto dominance $c_1 \succ_{\mathscr{P}} c_2$ holds if and only if weak dominance holds and there exist at least one preference category with a strictly higher preference value:

$$c_1 \succeq_{\mathscr{P}} c_2 \wedge \exists A_i : \pi_{A_i}(c_1) > \pi_{A_i}(c_2)$$

Weak Pareto dominance $\succeq_{\mathscr{P}}$ of configuration plan x over configuration plan x' occurs if $\pi_{Pcat}(x)$ weakly dominates $\pi_{Pcat}(x')$, which in turn occurs if and only if $\pi_{Pcat}(x)_i \geq \pi_{Pcat}(x')_i, \forall i \in \{1, ..., m\}$.

A set of configuration plans in which the values of their preference categories do not dominate each other, is called a *Pareto Set* of configuration plans. It can be a weak Pareto set, or a strict Pareto set, depending on the type of Pareto dominance used. Unless otherwise stated, the dominance relation for the Pareto set refers to $\succ_{\mathscr{P}}$.

Definition 11 C3PR$_{\mathscr{L}}$ (Lorenz) Problem

A C3PR Lorenz problem C3PR$_{\mathscr{L}}$ is a C3PR problem in which the dominance relation $\mathscr{R}_d$ is the Lorenz dominance relation. Just like with Pareto, there exists weak Lorenz dominance $\succeq_{\mathscr{L}}$ and strict Lorenz dominance $\succ_{\mathscr{L}}$. The Lorenz dominance relations are partial order relations such that Pareto monotonicity, Impartiality, and the Pigou-Dalton principle of transfers are satisfied. A set of configuration plans in which the Lorenz dominance relation applies, is called a *Lorenz Set* of configuration plans and we note that it is a subset of the Pareto set.

Let us consider $f_{(1)}(x)f_{(2)}(x)...f_{(m)}(x)$ as the components of a preference value $f = (f_1(x), f_2(x), ..., f_m(x))$, *sorted by increasing order*. Let $x \in \mathbf{X}$ be a preference value. Let the generalized Lorenz vector associated to x be $\mathscr{L}(x) = (l_1, ..., l_m)$, where $l_1 = f_{(1)}(x), l_2 = f_{(1)}(x) + f_{(2)}(x), ... \, l_m = \sum_{i=1}^{m} f_{(i)}(x)$. Let $x, x' \in \mathbf{X}$ be two preference values. According to Nagy et al. [19] the value x weakly Lorenz dominates ($\succeq_{\mathscr{L}}$) the value x' if and only if: $\mathscr{L}(x) \succeq_{\mathscr{P}} \mathscr{L}(x')$. Similarly, the value x strictly Lorenz dominates ($\succ_{\mathscr{L}}$) the value x' if and only if: $\mathscr{L}(x) \succ_{\mathscr{P}} \mathscr{L}(x')$.

Unless otherwise stated, the dominance relation for the Lorenz set refers to $\succ_{\mathscr{L}}$.

To illustrate the difference between the two dominance operators we assume that we have plans with corresponding preference values given below:

$$a = \langle 0.5, 0.5, 0.5 \rangle \quad b = \langle 0.5, 0.4, 0.6 \rangle$$

$$c = \langle 0.5, 0.3, 0.7 \rangle \quad b' = \langle 0.5, 0.4, 0.7 \rangle$$

Here only $b' \succ b$, and the Pareto optimal set is $\{a, b', c\}$. The corresponding generalized Lorenz vectors are:

$$\mathscr{L}(a) = \langle 0.5, 1.0, 1.5 \rangle \quad \mathscr{L}(b) = \langle 0.4, 0.9, 1.5 \rangle$$

$$\mathscr{L}(c) = \langle 0.3, 0.8, 1.5 \rangle \quad \mathscr{L}(b') = \langle 0.4, 0.9, 1.6 \rangle$$

Hence the Lorenz optimal set is $\{a, b'\}$.

4 Solving C3PR optimization problems

In order to find solutions to C3PR optimization problems, we introduce Algorithm 1. The algorithm searches the space of partial solutions, and is based on partial order planning [21] [27]. An ordering function $\mathscr{H}$ is used to sort the partial solutions in the search array. The algorithm progresses in two stages, initially a heuristic function H_1 is used as $\mathscr{H}$ in order to find the first solution. Once the first solution has been found the dominance function $\mathscr{R}_d$ for the problem is used as $\mathscr{H}$ instead and is used to expand the search front along possibly optimal solutions and to prune non-optimal solutions. Step 8 uses the traditional partial ordered planning resolution of conflicts [27]. Steps 11 handles a missing requirement of a functionality by adding a new functionality (or reusing an existing one), and adding a link from the latter to the former. In partial order planning, only causal links are added, but here also information links can be added. Notice that the latter implies simultaneous execution of the linked functionalities.

The solution to a C3PR optimization problem is the maximum set of non-dominated fully admissible configuration plans (FACP), which hints at the importance of the dominance operator. Currently this algorithm supports solving C3PR$_{\mathscr{P}}$ and C3PR$_{\mathscr{L}}$. We note that solutions to C3PR$_{\mathscr{L}}$ are subsets of C3PR$_{\mathscr{P}}$, and our hypothesis is that C3PR$_{\mathscr{L}}$ prunes significantly more from the search and runs faster. To implement Lorenz dominance, we strictly followed the method proposed by Nagy et al [19].

For solving the C3PR problem instance, the algorithm keeps a search array with partial configuration plans, and an array of dominant FACP. The array of dominant configurations contains the set of FACP's that do not dominate each

Algorithm 1 Algorithm for solving C3PR optimization problems

Input: C3PR problem P_{C3PR}, first heuristic H_1, dominance func. $\mathscr{R}_d$.
Output: Maximal non-dominated set $\mathbf{S}_{P_{C3PR}}$ of solutions

1. Ordering function: $\mathscr{H} \leftarrow H_1$
2. Functionality instance: $f_g \leftarrow \langle I_g, C_g, \emptyset, \emptyset \rangle$ where I_g, C_g are the goal requirements.
3. Configuration plan: $C_i \leftarrow \langle \{f_g\}, \emptyset, \emptyset, \{\langle f_g, f_g, \mathscr{R}_= \rangle\} \rangle$
4. Search queue: $S \leftarrow \{\langle C_i, \mathscr{H}(C_i) \rangle\}$
5. Solution set: $\mathbf{S}_{P_{C3PR}} \leftarrow \{\}$
6. If S is empty, quit and return $\mathbf{S}_{P_{C3PR}}$.
7. Take $N_x = \langle C_x, h_x \rangle$ from search queue S.
8. If an effect of any $f_x \in \mathbf{F}_x$ of plans C_x in node N_x threatens any link returned by Lk_c of C_x :
   (a) generate child partial conf. plans from conflict resolution.
   (b) check children for consistency.
   (c) add all consistent children to S using ordering function $\mathscr{H}$
   (d) Go to 6
9. If no unsat requirements in C_x:
   (a) If $\mathbf{S}_{P_{C3PR}}$ is empty:
      i. $\mathscr{H} \leftarrow \mathscr{R}_d$
      ii. $h_x \leftarrow \mathscr{H}(C_X)$
      iii. Add N_x to $\mathbf{S}_{P_{C3PR}}$.
      iv. Recalculate Heuristic value for all elements in S using $\mathscr{H}$, pruning away those dominated by h_x
      v. Sort S
   (b) If $\mathbf{S}_{P_{C3PR}}$ is not empty:
      i. Prune from S all nodes dominated by h_x
      ii. Discard N_x if dominated by any element in $\mathbf{S}_{P_{C3PR}}$.
      iii. Prune from $\mathbf{S}_{P_{C3PR}}$ all nodes dominated by by h_x
   (c) Go to 6.
10. Select unsat req r_x of some $f_x \in \mathbf{F}_x$ of plan C_x
11. **for-each** instantiate or re-use f' with source r'_x compatible with r_x.
   (a) Create child plan $C' \leftarrow C_x$ where $C' = \langle \mathbf{F}', Lk'_i, Lk'_c, Ex' \rangle$
   (b) $\mathbf{F}' \leftarrow \mathbf{F}_x \cup \{f'\}$
   (c) If r_x is a causal requirement of f_x:
      i. $Lk'_c \leftarrow Lk_c \cup \{r_x \mapsto r'_x\}$
      ii. $\mathbf{Ex}' \leftarrow \mathbf{Ex} \cup \{\langle f', f_x, \mathscr{R}_{\text{before}} \rangle, \langle f', f_g, \mathscr{R}_{\text{before}} \rangle\}$
   (d) If r_x is an information requirement of f_x
      i. $Lk'_i \leftarrow Lk_i \cup \{r_x \mapsto r'_x\}$
      ii. $\mathbf{Ex}' \leftarrow \mathbf{Ex} \cup \{\langle f', f_x, \mathscr{R}_{\text{simultaneous}} \rangle, \langle f', f_g, \mathscr{R}_{\text{before}} \rangle\}$
   (e) If $\mathscr{H}(C')$ dominated by any value in $\mathbf{S}_{P_{C3PR}}$ discard C'
   (f) If $\mathbf{Ex}'$ consistent: insert $\langle C'_x, \mathscr{H}(C'_x) \rangle$ into S
   (g) Go to 6

other. The planner first starts with a heuristic as the ordering function $\mathscr{H} = H_1$. Once the first FACP has been found, a one-time event 9(a)i is triggered in which $\mathscr{H} = \mathscr{R}_d$ and the ordering value is recalculated for every element in the search array (into a preference-based score). For every FACP found (including the first one) a pruning phase is triggered both in the search and in the dominant FACP array. All configurations dominated by this FACP are removed from the search array and the array of dominant FACP.

When preferences are used for comparing configurations, we combine the preference value of the sources of all the links in the configuration plans as given by Definition 7. This provides a monotonically decreasing preference value, necessary for correctness of step 9(a)iv and 11e.

5 Experiments and Preliminary Results

The goal of the experiments is to obtain a preliminary idea of the performance of the planner as well as to establish how different types of problems affect the performance of the planner, considering Pareto and Lorenz dominance. The main null hypothesis for these experiments is that the planner is equally sensitive to the variation of the preliminary set of parameters. A secondary null hypothesis is that the planner has the exact same performance when running with Pareto and with Lorenz dominance. The experiments are intended to refute both hypotheses. The results here presented are intended to show the community that it is possible to have a fast planner for solving problems of configuration planning with multiple partially ordered preferences.

In order to setup the experiments we have taken a statistical approach [24] where we calculate the mean value of properties of the algorithm over *all* problem instances up to a given size and with a few other limitations outlined below.

The first step for setting up the experiments was to define the population over which we can ensure a statistically significant measurements. In practice, this means defining the set of parameters used for problem generation and splitting it into a set of *families* of problems where a describing factor of a family is eg. number of functionalities in the problem. For the remaining parameters, eg. the specific functionalities used, we have sampled using a random uniformly distributed sampling over the population set. For this we have made exploratory initial tests for obtaining the behavior of different parameters.

To generate the problems we have generated a dictionary of state variable names and values for state variable assignments. Based on this we have randomly generated a number of functionalities, and for each functionality generated a random number of requirements and sources. The domain function for each functionality **D** has been built to accept exactly one state variable name or assignment for each requirement or source of the functionality. In practice this means that no unification or other correlation between variables have been allowed in these tests. Finally the preference functions for each functionality have been randomly generated to give values in a preference universe $\mathbf{U}_{pref}$ consisting of exactly three preference categories.

The parameters for the different families of problems investigated are:

- Average # of requirements (2, 4, 6), sources (2, 4, 6), goals (2, 3, 4), available instantiations in the functionalities (2, 5, 10) and number of functionalities (64, 128, 256).
- Size of the dictionary used to create sources and requirements (16, 32, 64 for 64 functionalities, 32, 64, 128 for 128, and 64, 128, 256 for 256)

– # of functionalities that only contain sources (10, 25 and aprox. 50 % of each number of functionalities)

A second practical problem is to define what is a reasonable time to wait for a solution since a few individual problem instances may otherwise block the experiments almost indefinitely, and for all practical purposes we should consider problem instances that take unreasonably long to solve as non-solvable by the algorithm. We note that the planning problem has an exponential growth in the time consumption and in the memory consumption measured by the size of the found solutions. Furthermore, the practical implementation of the algorithm is mainly limited by the amount of available computer RAM memory which can be exhausted in less than a minute. Therefore we define a cut-off for tractability by maximum consumed memory so that we can avoid excessive garbage collection and swapping.

The results here presented have a confidence level of 1σ for N=35, with a small percentage of the standard deviation in confidence levels, meaning approx. 14000 experiments per planning approach according to method 2 in [24], for each of the ca. 2200 families of problems.

The authors are using the data here presented as preparation for a third round of experiments in 3σ, in order to identify which parameters of interest affect how much harder the planning problems become. Then, the parameters with marginal influence are discarded. This is done in order to diminish dimensionality in the number of problem types to evaluate.

The variables of interest measured on every experiment include whether or not the run was terminated by full exploration, or dropped after a limiting condition was reached, as well as: the number of solutions found before termination, the number of generated, visited and pruned nodes, the maximum size of the search list, the number of generated configuration plans (differs from generated nodes in the fact that many configuration plans are pruned before getting to the search list of the algorithm), the time in nanoseconds for a run, and the Average branching factor of a run.

5.1 Preliminary Results

Table 1 presents the means and standard error for different variables of interest for Lorenz and Pareto. In order to obtain this means, for each problem family we partitioned results into blocks of 35 samples, considering only individual runs terminated by full exploration. If a block contained less than 35 samples, it was discarded from this calculation. The means for each block were calculated, and then the result in this table is the mean of the means of all blocks. The standard error is calculated by dividing the sample estimate of the standard deviation by the square root of the size of the samples ($\sqrt{(35)}$). Table 2 presents the means of the problem families with the smallest and largest means, for each variable of interest, for Lorenz and Pareto.

Table 1: Total Sample Means and Standard Error for variables of interest, for Lorenz and Pareto dominance

Variable of interest	**Lorenz** StdErr	**Lorenz** Mean	**Pareto** Mean	**Pareto** StdErr
Time(s)	0.0624	0.159	0.102	0.0226
Generated nodes	968.15	6935.12	7009.66	974.71
Visited nodes	170.01	1016.96	1029.79	171.45
Pruned nodes	844.42	6053.49	6129.69	852.48
Generated configurations	964.672	6499.147	6487.437	904.303
Maximum size of searchlist	499.540	3965.122	3939.596	459.094
Avg. Branch factor	0.345	7.075	7.292	0.378
Solutions found	2.947	15.249	17.350	4.178

Table 2: Min and Max Sample Means for variables of interest, for Lorenz and Pareto dominance

Variable of interest	**Lorenz** MinMean	**Lorenz** MaxMean	**Pareto** MinMean	**Pareto** MaxMean
Time(s)	0.0038	20.81	0,0033	4,67
Gen. nodes	47.0	51749.03	45.23	62299.91
Visited nodes	15.6	8490.83	15.6	8724.51
Pruned nodes	7.34	45050.6	7.34	55204.54
Generated configs	226.514	44961.886	279.229	46129.114
Max size of searchlist	146.971	13641.0857	162.8	11983.857
Avg. Branch factor	1.63	12.837	1.59	13.121
Solutions found	2.057	424.543	2.20	481.886

The first important observation from Tables 1 and 2, is that the average time for solving C3PR problems within the tested parameter ranges is quite reasonable for most practical purposes. If the reader is interested on accessing the sources of the implementation, they are available via bitbucket (for access, please contact the first author).

The second important observation from Table 2, is that the planner is evidently not equally sensitive to the variation of the preliminary set of parameters, which refutes the main null hypothesis of these experiments.

From observing Table 1, we can observe that for Lorenz and Pareto, the means of the variables of interest are within the standard errors of both. When putting Table 1 in perspective with Table 2, we observe a significant difference between the Min and the Max means. Such difference shows that the problem instances can have a significant effect on the performance of the planner, and suggests that a more balanced standard error can be achieved if the total are calculated considering clusters of families of similar behavior. As mentioned in Section 5, this is one of the analysis that

will be performed in order to diminish dimensionality in the run parameters for the 3σ tests. Therefore, the secondary null hypothesis could not be refuted. Further tests and analysis are necessary in order to confirm or refute the secondary hypothesis.

6 Conclusions

In this paper we have presented a representation, a language and an algorithm for configuration planning with partially ordered quantitative preferences, inspired in partial order planning [21] [27] and in the soft constraints framework from Bistarelli et al [6]. Moreover, we have presented preliminary results on a number of variables of interest, for a number of families of problems and for two dominance operators (Pareto and Lorenz).

These preliminary results show that the existing planning approach is suitable for practical applications falling within the types of problem families tested, and that is possible get significantly different performance for different families of problems. The results can be used as a benchmark for similar experiments regarding configuration planning with preferences.

7 Acknowledgement

This research was supported by the GiraffPlus EU Project, funded by the European Community's Framework Programme Seven (FP7) under contract #288173.

References

1. James F Allen and Johannes AGM Koomen. Planning using a temporal world model. In *IJCAI*, volume 8, pages 711–714, 1983.
2. Jorge Baier, Fahiem Bacchus, and Sheila A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173(5-6):593–618, 2009.
3. Jorge A. Baier and Sheila A. McIlraith. Planning with preferences. *AI magazine*, 4(29):25–36, 2008.
4. M. Bienvenu, C. Fritz, , and S. A. McIlraith. Planning with qualitative temporal preferences. In *Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR-06)*, pages 134–144. AAAI Press, 2006.
5. M. Bienvenu, C. Fritz, , and S. A. McIlraith. Specifying and computing preferred plans. *Artificial Intelligence*, 175(7–8):1308–1345, 2011.
6. S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based csps and valued csps: Frameworks, properties, and comparison. *Constraints*, 4(3):199–240, 1999.
7. Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artifcial Intelligence Research (JAIR)*, 21:135–191, 2004.
8. R. Brafman and Y. Chernyavsky. Planning with goal preferences and constraints. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 182–191, 2005.
9. Ronen I. Brafman and Yuri Chernyavsky. Planning with goal preferences and constraints. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS-2005)*, 2005.
10. Ronen I. Brafman, Carmel Domshlak, and Solomon E. Shimony. On graphical modeling of preference and importance. *Journal of Artificial Intelligence Research*, 25:389–424, 2006.
11. Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1):61–95, 1991.
12. Coradeschi et al. Giraffplus: Combining social interaction and long term monitoring for promoting independent living. In *6th International Conference on Human System Interactions (HSI)*, 2013.
13. Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.
14. Alfonso Gerevini and Derek Long. Preferences and soft constraints in pddl3. In *ICAPS workshop on planning with preferences and soft constraints*, pages 46–53, 2006.
15. Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.
16. Christophe Gonzales, Patrice Perny, and J Ph Dubus. Decision making with multiple objectives using gai networks. *Artificial Intelligence*, 175(7):1153–1179, 2011.
17. G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.
18. Robert Lundh. *Robots that Help Each Other: Self-Configuration of Distributed Robot Systems*. PhD thesis, Örebro University, School of Science and Technology, 2009.
19. Réka Nagy, Mihai Suciu, and D Dumitrescu. Exploring lorenz dominance. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on*, pages 254–259. IEEE, 2012.
20. Lynne E Parker and Fang Tang. Building multirobot coalitions through automated task solution synthesis. *Proceedings of the IEEE*, 94(7):1289–1305, 2006.
21. J. Scott Penberthy and Daniel S.Weld. Ucpop: A sound, complete, partial order planner for adl. In *Proceedings of the third international conference on knowledge representation and reasoning*, pages 103–114. Citeseer, 1992.
22. A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–57. Institute of Electrical and Electronics Engineers, 1977.
23. Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.
24. Lia Susana d.C. Silva-Lopez and Mathias Broxvall. Empirical methods for evaluating properties of configuration planning algorithms. In O'Grady et al, editor, *Evolving Ambient Intelligence*, volume 413 of *Communications in Computer and Information Science*, pages 114–119. Springer International Publishing, 2013.
25. T. C. Son and E. Pontelli. Planning with preferences using logic programming. *Theory and Practice of Logic Programming*, 6(5):559–607, 2006.
26. Mark Vilain and Henry Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 377–382, 1986.
27. Haakan L. S. Younes and Reid G. Simmons. Vhpop: Versatile heuristic partial order planner. *J. Artif. Intell. Res. (JAIR)*, 20:405–430, 2003.