SEVENTH FRAMEWORK PROGRAMME	**Project Acronym: GiraffPlus** **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**

D 3.1 Context Inference and Configuration Planning Prototypes

WP related to the Deliverable:	WP3 Intelligent Monitoring and Adaptation Services
Nature:	Prototype
Dissemination Level :	Public
Version:	V1
Author(s):	Lars Karsson, Lia Silva-Lopez, Jonas Ullberg, Federico Pecora
Project Participant(s) Contributing:	Lars Karsson, Lia Silva-Lopez, Jonas Ullberg, Federico Pecora
Contractual Date of Delivery:	2012-12-31
Actual Date of Delivery:	2013-02-01

Document History

Version	Date	Type of editing	Editorial
V1.0	2013-02-01	First complete version	

Deliverable Summary

In this report we present the results after the first year in WP3 regarding Intelligent Monitoring and Adaptation Services, which consists of two major components: context recognition and configuration planning. Prototypes of these two components have been implemented, and those prototypes constitute the main focus of the report. Regarding the context recognition part, the main contribution is a novel algorithm for performing inferences in the context of missing data points in a time line. The capabilities of the context recognition match the relevant requirements from WP1. The configuration planner is inspired by partial-order task planning techniques, but supports both information and causal dependencies between functionalities. We also address integration, deployment and demonstration.

Contents

1 Introduction

In this document we present the results after the first year in WP3 regarding Intelligent Monitoring and Adaptation Services, which consists of two major components: context recognition and configuration planning. Prototypes of these two components have been implemented, and those prototypes constitute the main focus of the report. However, we also address integration, deployment and demo.

The report is organized as follows:

Section 2 recapitulates the tasks of WP3 as stated in the Description of Work (DoW) that were active during the first year.

Section 2 presents the relevant concepts of Intelligent Monitoring and Adaptation Services as defined in deliverable D1.3 System Reference Architecture.

Section 4 recapitulates the component's part in the system reference architecture and the use cases as presented in deliverable D1.3.

Section 5 presents the prototype of the context recognition component, and in particular a novel technique for robust inference in the presence of gaps in the data. It also investigates how the capabilities of the context recognition compares to the requirements from deliverable D1.3.

Section 6 presents the prototype of the configuration planning component.

Section 7 presents the initial integration of the two components.

Section 8 presents the deployment of configurations.

Section 9 explains the parts of the context recognition and configuration planning components in the first-year demonstration (deliverable 5.2).

Section 10 concludes what has been achieved and what remains.

The following documents/deliverables are relevant for the present one. **Deliverable D1.3 System Reference Architecture** relates the user requirements presented in deliverable **D1.1 User Requirements & Design Principles Report** to the three main components of the system namely: Physical environment and Software infrastructure: (WP2: middleware, sensors, and Giraff robot); Intelligent Monitoring and Adaptation Services (WP3: context recognition module and configuration planner); and Data Visualization, Personalization ad Interaction Services (WP4: general data store, personalization module and interaction module). The user requirements are then translated into technical requirements considering the results of the deliverable **D1.2 Technological Components Specification**. The technical requirements are directing the architectural structure and the development of the system. As mentioned before, the present document recapitulates the architecural decisions from D1.3 that are relevant for the

Intelligent Monitoring and Adaptation Services, as well as reiterates some of the conceptual definitions from that document.

Deliverable 5.2 Video-milestone 1 is a video demonstration at the Ängen test site recorded in December 2012. It involves a scenario that illustrates many aspects of the GiraffPlus system, including context recognition and configuration planning.

2 WP3 tasks and deliverables during year 1

There were four active tasks in WP3 during the first year. These tasks will continue during year 2 and 3. There was also one deliverable, of which this report is a part. The tasks and deliverable are summarized below (complete versions are available in Appendix A).

Task 3.1 Long-term context recognition (OrU)

Context recognition must provide two capabilities in order to guarantee that relevant events/activities and trends are recognized in a timely fashion. First, data received from the sensor network needs to be interpreted and represented in symbolic form in order to allow inference mechanisms to assess whether specific conditions hold. This equates to bringing signal-level data (e.g., the readings of a proximity sensor, or raw physiological data) to a higher level of abstraction that indicates crisp states of the human being and of the environment. The second step is to actually perform inference in this information in order to obtain high-level descriptors of the state of the human user which can support human decision making (e.g., activities of daily living). The former problem of bridging the gap between signal-level data and symbols will be addressed using data-driven techniques like Hidden Markov Models, Neural Networks and other Machine Learning methods. The second step will be achieved through the use of constraint-based reasoning techniques. For instance, we want to be able to use information about the assisted person entering the kitchen (a primitive event which can be detected by sensors after some processing) as part of an inference that the user is having lunch (abstract activity).

Work done during year 1: The focus has been on extending previous techniques in order to tackle the problem of brittle inference by admitting many interpretations of sensor timelines. The point is that with previous techniques, a single point of missing data could invalidate an inference extending over a long interval. Allowing timelines where such gaps are filled in can enable inferences under such circumstances. We consider this to be a very important feature in order to perform robust long-term monitoring in an environment where disturbances are likely. A first propotype of the system is presented in section 5.

Task 3.2 Configuration planning for efficient activity recognition (OrU)

Given a certain set of activities and a set of available sensors and programs for extracting information from these sensors, the sensor network needs to be configured in such a way that it is capable of of inferring the activities from the sensor data. Such a configuration specifies what sensors to be used, and what programs will be used to extract information at different levels of abstraction from them. For this purpose, we will develop configuration planning techniques, that can automatically find an optimal configuration for a given task (i.e. given set of activities of interest). The configuration planner can also contribute to robustness by automatically repairing configurations if a sensor fails or is removed.

Work done during year 1: A first prototype of a configuration planner has been developed. It is based on flaw repair, i.e. goals and subgoals as well as potential inteferences are considered flaws that can be repaired by extending or constraining the configuration. This makes ithe planner suitable not only for planning from scratch, but also for adding new goals or repairing failing parts of a configuraion.

Task 3.3 Integration of context inference and configuration planning (OrU)

This task will focus on the integration of configuration planning into the constraint-based context inference framework developed in task 3.1. First, the ability to specify hierarchical decompositions will be built into the constraint reasoning system. This will allow complementing the constraint language used for specifying correlations between sensor readings and human state with relevant information on which sensors should be activated and how they should be configured to maximize the information gathering process. Secondly, work will focus on developing hierarchical reasoning techniques on top of the temporal inference mechanism of the context inference and actuation infrastructure.

Work done during year 1: An initial intergration has been set up, and a protocol has been established. We expect the protocol to be extended considerably in the future.

Task 3.5 Deployment (OrU, CNR-ISTC, MDH)

In connection with the integration tasks of WP5, this task aims to design and implement routines for taking a description of a configuration generated by the configuration planner and start and connect the actual sensors, devices, interfaces and other programs specified in that description. In addition, failures in components and communication as well as successful task completions should be detected and responded to by reconfiguration.

Work done during year 1: a fairly large engineering effort has been put into this task, in order to make the configuration planning and context recognition systems to work together with the sensor network and access and store the relevant data.

D3.1: Context inference and configuration planning prototypes

Initial prototypes of context recognition and configuration planning systems. A report will describe the prototypes. [month 12]

3 Concepts

The Intelligent Monitoring and Adaptation Services consider the environment abstractly as a collection of **state variables**, which at each point in time have some values. These state variables may represent for instance:

- Aspects of the layout of the apartment (rooms, furniture, etc.)
- Position and motion of individuals in the apartment (e.g. person1 in bedroom, on bed, not moving)
- Position and motion of various items in the apartment (e.g. book1 on table in bedroom, giraffe robot in living room)
- Status of various items in the apartment (e.g. light on/off, machine on/off, door open/closed)
- Various physiological parameters of the individuals in the apartment (e.g. heart rate of person1, weight of person1)

A state variable may be static or dynamic. It may be observable directly through sensors, or it may be indirectly derivable from other state variables (at least in some circumstances). Some state variables may be controllable, i.e. they can be changed in a predictable manner by the sensor network.

The **context recognition system** reasons about state variables and activities. An **activity** has an extension in time, and it typically involves changes in state variables (e.g. eating-dinner, sleeping, food-cooking, robot-moving). The context recognition system maintains time lines with state variables and activities. The time line is a partial mapping of state variables and time to values and from activities and time to { on,off }. The time line is also stored in the database for long-term access. The context recognition system utilizes context models, including context rules for deriving activities from state variables and other activities. These derived activities may involve actuation, e.g. in the form of an alarm.

The **sensor network**[1] is considered as a collection of devices (sensors and possibly also actuators, including those mentioned in Table 9 in D1.3) which can be configured and connected in different ways. Each sensor can potentially obtain the values of one or more of the state variables, depending on the current state

[1]Work package 2 is concerned with four components: sensor networks, middleware, data storage and Giraff robot. The physical environment of the primary user is sensed and modified by sensor networks and actuators. The middleware abstracts the sensor network as services that will be provided at higher layers (e.g. context recognition) in terms of topics following a publish/subscribe pattern. The data retrieved from sensors and the events triggered by the upper layers will be stored in a data storage system for further and future processing following the same pattern. (From deliverable 3.1)

and on how the sensor is configured. Sometimes, a state variable is not directly obtainable, but it has to be derived from the data from one or more sensors in the network. Sometimes, some other state variable may need to be changed in order to obtain a certain state variable. An actuator can alter one or more state variables. There are also purely computational processes (algorithms) for processing sensor data at various levels of refinement (or producing control) as part of the sensor network. A **functionality** is a program that either operates directly on a sensor or actuator, or processes data from and/or delivers data to other functionalities. Note that a sensor or actuator can realize several functionalities, depending on what program is connected to it. Functionalities can communicate with each other through their inputs (of different types and contents) and outputs. They may also require certain state variable assignments to function properly (e.g. a camera may require the light to be on), and/or they may modify certain state variables (i.e. switch on the light).

A **configuration** of the sensor network consists of:

- What functionalities are active and how they are configured.
- How outputs and inputs of functionalities are connected to each other, by using the subscription mechanism of the middleware.

The configuration (together with the current state) determines what state variables are monitored.

The task of the **configuration planner** is to configure the sensor network in terms of subscriptions (and possibly also giving parameters to functionalities) so that the state variables requested by the context recognition system are monitored. It also directs this data to the context recognition system. The configuration planner should also be able to adapt to changing conditions, e.g. functionalities added, removed, or malfunctioning, or changes in certain state variables (e.g. light =off). The configuration planner may produce configurations that change over time, in particular when actuation (pre- and postconditions) is involved.

The context recognition system interacts with the **personalisation services** developed in WP4[2]. There are multiple instances of the services, relating to different secondary users with different needs. Each of them can provide:

[2]The Data Visualization, Personalization and Interaction Service is the part of the architecture responsible for creating user-oriented and personalized services. More specifically, the module provides to the different end-users suitable interaction modalities and specific intelligent services. The module can be broadly described as constituted by a Front-End, i.e., an Interaction Service in charge of providing all the human users with basic GiraffPlus interfaces and a Back-end responsible for providing intelligent services and personalization capabilities. The back end in turn is composed of the User Oriented Services module and Personalization Services module. The former is responsible for providing some proactive/reasoning services to the users, while the latter is in charge of maintaining users' profiles so as to tailor both services and interaction to the involved specific persons. (From deliverable 3.1.)

- Requests for state variables and activities, including time constraints (e.g. movement activities during night time, heart rate measurement once every morning).
- Selections of rules in context model for deriving activities.

Given the requests from these services, the context recognition system determines what state variables need to be monitored and requests these variables from the configuration planning system. It then continuously receives data about these state variables from the sensor network and/or the data storage, enters them into the time line and derive activities from them. The activities are in turn entered into the time line and so on.

4 WP3 in system reference architecture

During the work on deliverable D1.3, the structure in Fig 1 for the Intelligent Monitoring and Adaptation Services was worked out. In addition, a number of use cases based on the user requirements established in D1.1 were elaborated. These are presented in Fig 2, and the typical activities in the use cases labeled "visualisation/personalization and user" are presented in Fig 3.

In the rest of this section, we recapitulate the conceptual description of Intelligent Monitoring and Adaptation Services from D1.3.

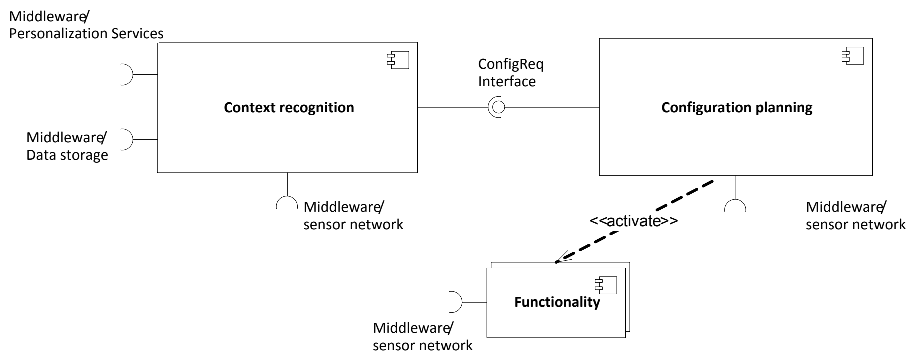


Figure 1: UML diagram of Intelligent Monitoring and Adaptation Services from the system reference architecture document

4.1 Interfaces

Most of the communication occurs through the middleware, as described in section 3.2.1 of deliverable 1.3. However, there are specialized protocols for communication with other components on top of that.

- Sensor network: the configuration planner can make requests for subscriptions of data, and the context recognition can receive data using those subscriptions.
- Personalization services: the context recognition can receive requests for activities to be monitored, and requests to select specific inference rules, or modify/create inference rules. It can also provide the personalization services with information about what activities and inference rules are available, or selected.
- Data storage: the context recognition can store and retrieve timeline data. As language, we will use the language JSON (JavaScript Object Notation, see `www.json.org`).

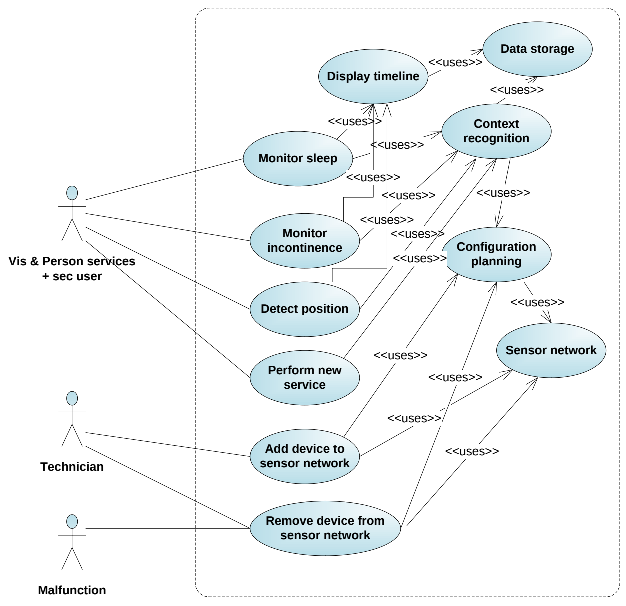


Figure 2: Use cases for Intelligent Monitoring and Adaptation Services

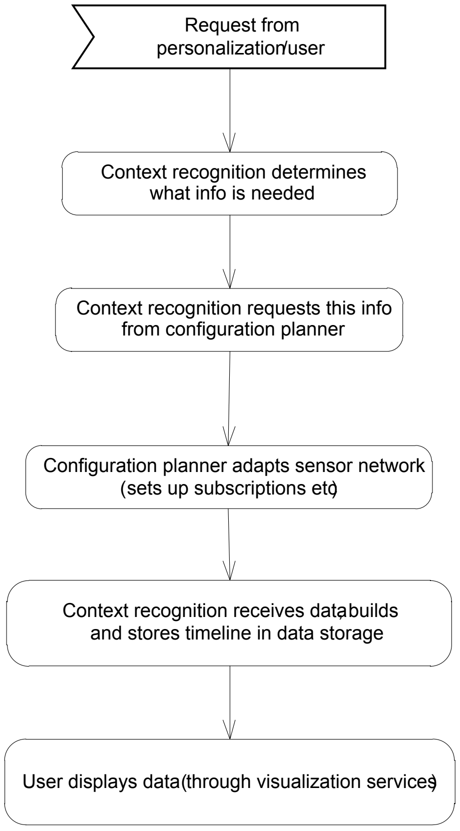

Request from personalization/user
Context recognition determines what info is needed
Context recognition requests this info from configuration planner
Configuration planner adapts sensor network (sets up subscriptions etc)
Context recognition receives data, builds and stores timeline in data storage
User displays data (through visualization services)

Figure 3: Activities in use cases ("visualisation/personalization and user") for Intelligent Monitoring and Adaptation Services

5 Context recognition

The aim of task 3.1 in GiraffPlus is to construct a system for inferring context based on a given model of how this context correlates to sensor traces. In particular, the system is to infer human activities from a sensor network in an apartment. Recall that secondary users of the GiraffPlus system, could via the Personalization Services request what activities to monitor. The context recognition recognizes those activities from the values of observable state variables and stores the result in the data storage. It is supported by the configuration planner (which is presented in the next section) in determining which of the available sensors to use in order to observe the required state variables.

This and other applications benefit from the ability to specify the model based on which context is inferred in a flexible and compact way. An easily specifiable model would allow, for instance, to easily configure a context recognition system to infer human activities and situations on a per-user and environment basis. Furthermore, the system should be able to be configured by a person without extensive knowledge of the underlying algorithm.

For this purpose we have found it useful to represent the states of the sensors and the inferred activities as intervals on different timelines. The model that describes the causal relationships between the states of the sensors and the inferred activities is provided as a set of quantitative Allen's interval algebra constraints. These constraints are posted between intervals representing sensor readings. This approach was first described by Ullberg et al. [2009] and then subsequently extended by Pecora et al. [2012].

In addition we tackle the problem of brittle inference by admitting many interpretations of sensor timelines, as has been reported in Ullberg et al. [2012] (from which much the text comes). This is achieved by performing temporal inference on multiple intervals contextually. That is to say, each sensor reading is represented as a *set* of flexible temporal intervals rather than only one. In order to assess whether temporal constraints hold among sets of intervals, we define a new temporal constraint propagation algorithm. This algorithm constitutes the basis of an abductive inference system which decides the overall context recognition problem.

5.1 Context recognition modules

The context recognition system consists of several independent modules, each handling its own specific task. The most important modules are the preprocessing, inference and extraction modules.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<config>

 <preproc name="discont" in="livingroom_couch.ir" out="in_livingroom" lim="20"/>
 <preproc name="discont" in="bedroom_door.ir" out="in_bedroom" lim="20"/>
 <preproc name="discont" in="kitchen_microwave.ir" out="in_kitchen" lim="20"/>
 <preproc name="discont" in="bathroom.ir" out="in_bathroom" lim="20"/>
 <preproc name="discont" in="bedroom_door.light" out="bedroom_light" lim="20"/>
 <preproc name="ttf" in="bed-presence" out="in_bed" lim="0"/>
 <preproc name="discont" in="kitchen_microwave.on" out="cooking" lim="20" thresh="0.5"/>
 <preproc name="discont" in="livingroom_tv.on" out="tv_on" lim="20" thresh="0.5"/>
 <preproc name="average" in="livingroom_couch.occupied" out="in_couch" lim="0.99"/>

 <rule out="watching_tv">
  <constraint from="watching_tv" type="during" to="tv_on" />
  <constraint from="watching_tv" type="during" to="in_couch" />
  <constraint from="watching_tv" type="contains" to="in_livingroom" />
 </rule>

 <extractor name="max" in="in_livingroom" out="in_livingroom" />
 <extractor name="max" in="in_bedroom" out="in_bedroom" />
 <extractor name="max" in="in_kitchen" out="in_kitchen" />
 <extractor name="max" in="in_bathroom" out="in_bathroom" />
 <extractor name="max" in="bedroom_light" out="bedroom_light" />
 <extractor name="max" in="tv_on" out="tv_on" />
 <extractor name="max" in="in_couch" out="in_couch" />
 <extractor name="max" in="watching_tv" out="watching_tv" />
 <extractor name="max" in="cooking" out="cooking" />
 <extractor name="max" in="in_bed" out="in_bed" />

</config>
```

Figure 4: Example of rules for preprocessing, inference and exteaction, in XML format.

5.1.1 Preprocessing module

The preprocessing module's responsibility is to fetch samples from the sensor network and use these samples to build a higher level representation of the events that takes place in the home. For instance, this might include taking a timestamped series of temperature readings and "down-sampling" these to a single or a set of temporal intervals that represent that the temperature has been **High** during the corresponding periods of time. Knowing that the temperature is **High** somewhere might now be directly meaningful to the caregiver, in which case this information is never displayed to the user but instead reasoned upon by the inference module. However, in some cases this module is able to provide meaningful user data on its own, for instance, it can also "infer" the times in which the person is lying in bed given a series of samples taken solely from a bed-pressure sensor. In Figure 4, some examples of preprocessing rules are given.

The preprocessing module fetches samples from a remote database. Since this is the only module that uses the samples directly, this feature allows the entire context recognition system to be restarted (with modified rules for instance) without loosing any data. In addition, fetched data is stored locally (encrypted) in a temporary location to limit the load on the database server, requiring only incremental retrieval of new data.

Note that preprocessing of data can also be performed by programs invoked by the configuration planner. How to best distribute preprocessing in order to get a flexible and efficient system is a matter of future investigation.

5.1.2 Inference module

The inference module infers activities by performing temporal constraint propagation on the domains of intervals generated by the preprocessing module. The current constraints consists of quantified versions of the ones found in Allen's Interval Algebra [Allen, 1983], however, the overall architecture supports the more expressive INDU algebra [Pujari et al., 2000] which adds constraints on the relative duration of intervals. For instance, this module can 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. However, after taking a shower the warm-water pipe remains warm for several hours, thus it combines this data with location data also provided by the preprocessing module to get a better estimate of the exact duration of the shower. In this hypothetical case, a rule could state that an activity, **Showering** occurs `During` **In_Bathroom** and `Contains` **Temperature_High_Shower**. For more examples, see Figure 4.

The extraction module's responsibility is to generate timelines that can be visualized to the users. 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. Currently this module only supports one type of extraction method, which simply extracts the maximum duration interval for an activity. In the system this information is readily available which contrasts with the use of, for instance, chronicle recognition in which an additional propagation step is required to determine the same information. (The most common solution in that case would be to extract the earliest start-, earliest end-time.) For examples, see Figure 4.

In the reminder of this section, we focus on context inference and in particular in the presence of gaps in the data.

5.2 Representation

We illustrate the context recognition problem by giving a simple example of how inference is done from Pecora et al. [2012]. In this work, intervals are generated from rich sensory data that is provided by several sensors in a home environment. The goal is to provide a higher-level representation of what is happening in the world that is rich enough to reason about, but is unencumbered by unnecessary detail. Each generated interval represents a fact that holds true during a limited period of time, for instance that the humidity in the bathroom was **high** between 14:00 and 14:15.

In order to infer context, the generated set of intervals are combined with a model consisting of rules of the form:

$$\textbf{Cooking}\ \texttt{Equals}\ \textbf{Stove} \wedge \textbf{Cooking}\ \texttt{During}\ \textbf{Kitchen}.$$

Such rules define how context, in this case the activity of **Cooking**, can be

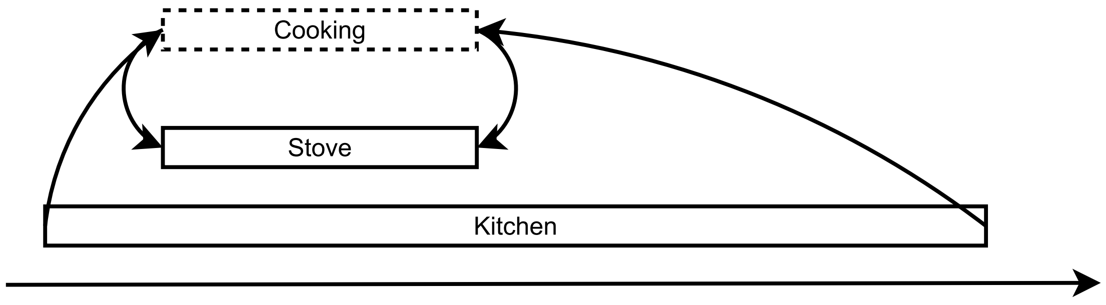


Figure 5: An example of a pattern in the sensory data that allows an activity to be inferred.

inferred from intervals representing sensed data. Rules like these define abstract patterns of constraints in Allen's interval algebra [Allen, 1983] that should be satisfied in order to recognize an activity. The inference itself is done by iteratively trying to constrain a Simple Temporal Problem (STP) [Dechter et al., 1991] in which the intervals representing sensed data and inferred activities are managed as pairs of timepoints, representing the start and the end time of the interval in question.

The high level Allen interval constraints are represented as simple distance constraints between timepoints in the STP. In the example above, the two constraints reference 6 timepoints in the STP; the start and the end times of the three intervals **Stove**, **Kitchen** and **Cooking**. A constraint such as

Cooking `Equals` **Stove**

is represented by two simple distance constraints in the STP that constrain the start and end times of the **Cooking** and **Stove** intervals so that they can only take on the same values. During inference, each combination of intervals that are referenced by such a pattern must be tried or pruned away by a search procedure. For instance the **Stove** might have been turned on several times in the past, and the person has been in the **Kitchen** at multiple times in the past. In this case, each combination of choices from these two groups of intervals have to be evaluated. Each possible combination is tried by propagating a STP, and if the STP has a consistent solution the pattern is considered satisfied, and the **Cooking** activity is thus inferred.

Figure 5 shows a scenario where the **Cooking** activity is successfully recognized.

Formally, our context recognition problem can be described as a constraint satisfaction problem (CSP) Tsang [1993] 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 interval either denotes that a sensed fact holds

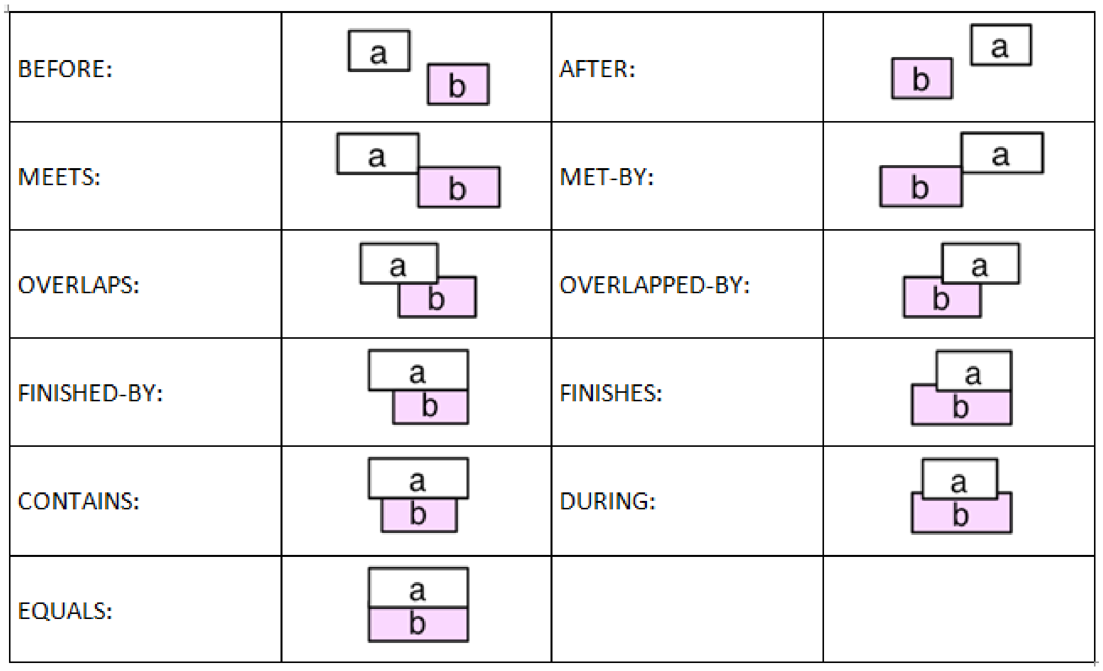


Figure 6: Primitive relations of the Allen algebra.

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

$$c_j = \{(v_0\ R_1\ v_1) \wedge (v_0\ R_2\ v_2) \wedge \ldots\}$$

where each c_j constrains the domains of two variables. v_0 is a variable representing the timeline of an inferred activity, 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 directed acyclic graph. The nodes at the very bottom of that graph (i.e. those with no arcs starting from them) represent sensed variables which in the GiraffPlus system will be observed by using sensors. These latter variables will be constrained by the actual sensor readings up to the present point in time.

Each R_i is a quantitative binary Allen relation, i.e. a disjunction of the primitive Allen relations in Figure 6. In addition, the relations may be followed by one or more metric bounds of the form $[l, u]$, depeding on the type of relation. For instance, a `Before` $[2, 4]$ b means that a must end between 2 and 4 time units before b begins. The relations defines temporal relations between the variables that should hold in order for an activity to be inferred. 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 context recognition problem is the projection of a solution to the CSP on the variable representing the inferred

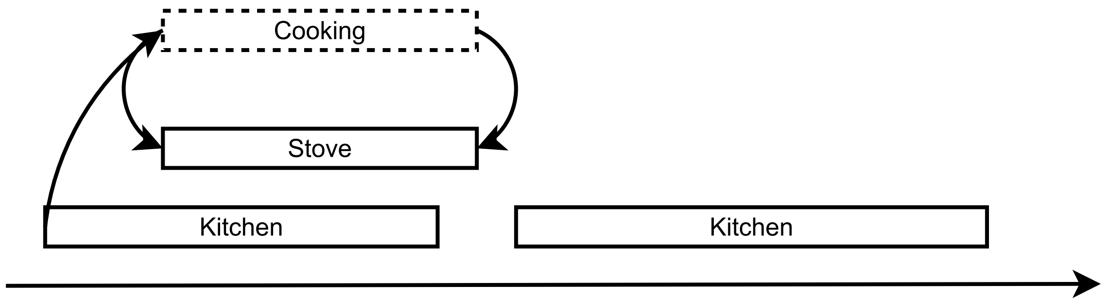


Figure 7: An example of a pattern in the sensory data that prevents an activity from being inferred.

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 sensed variable, 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.

5.2.1 Supporting robust inference in the presence of gaps

In prior work Ullberg et al. [2009], we found that although constraints taken from Allen's interval algebra are a convenient way to describe such relations, they are also very brittle in the sense that small deviations in how the raw sensory data is interpreted and placed on the timelines can prevent activities from being inferred. One possible way of overcoming this is through the use of fuzzy Allen's interval constraints [Mansouri, 2011]. In this work, constraint violations are allowed to some degree, thus enabling activities to be inferred with a "low likelihood" in case the model is not fully supported by the sensor readings. This approach, however, introduces problems when interpreting the inferred activities. Specifically, one has to provide a threshold on the likelihood of an inferred activity in order to determine a response to it, and this choice seems to lack rationale.

As earlier mentioned, we tackle the problem of brittle inference in the opposite way, that is, by using non-fuzzy, quantified Allen interval constraints, but instead admitting many interpretations of sensor timelines.

Figure 5 showed a potential scenario that could unfold where the **Cooking** activity is successfully recognized. In Figure 7 the same activity cannot be recognized due to the introduced discontinuity in the **Kitchen** timeline. The key

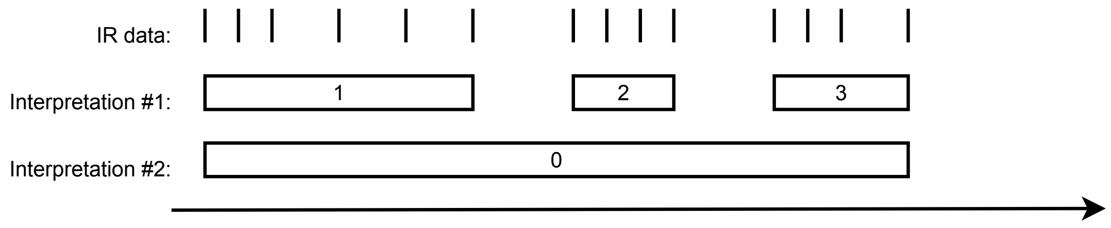


Figure 8: Two timeline representations of the same sensory data.

point is that even though these two scenarios 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 5 in mind, and small deviations from the optimal scenario can prevent us from recognizing the activity. This example is inspired by a real world deployment of sensors in an apartment and could easily arise when using a Passive Infrared (PIR) sensor for instance (a motion sensor often used in burglar alarms). This sensor is characterized by the fact that it emits a Boolean reading at regular intervals reflecting if movement has been sensed or not. When forming intervals out of these readings, we must interpret them to reflect if a person is in a room or not. This can be done, for instance, by allowing a fixed temporal window of discontinuities among consecutive readings indicating that movement has been registered. This situation is illustrated in Figure 8, which shows a set of Boolean readings indicating movement, and also two timelines that have been formed out of this data but with different thresholds of allowed discontinuity. In Interpretation #1, the translation of the discrete readings into intervals is quite strict so that discontinuities are easily introduced, whereas Interpretation #2 is resilient enough to only create one interval out of these readings.

The problem illustrated in Figure 7 could possibly be overcome by altering the rule so that **Cooking** is required to be `Overlapped-By` **Kitchen** rather than occur `During` **Kitchen**. This constraint would however not be satisfied if being in the **Kitchen** is first sensed after the **Stove** is turned on.

In order to overcome this problem we use 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. Thus, we would like to be able to reason on multiple *interpretations* of the same sensory data, each providing additional support for inferring an activity.

In the following sections we present a geometric representation of the domains of the variables in V. This representation allows us to define a propagation algorithm which achieves arc-consistency. As we will see, arc-consistency is complete under certain assumptions because of the tree structure mentioned above.

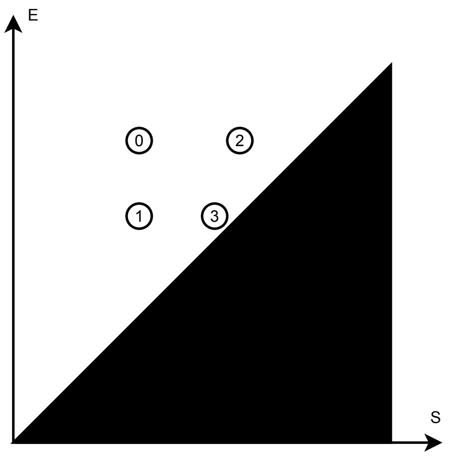


Figure 9: A 2 dimensional representation of a set of enumerated intervals. The S axis represents start times, and the E axis represent end times, and a point in the diagram represents a single interval with corresponding start and end time. The black area consists of impossible intervals that end before they start. The intervals from 8 are indicated.

5.2.2 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 Interpretation #1 and #2 in Figure 8. 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 context recognition which rely on an explicit representation of each interval 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 2 dimensional graph as in Figure 9. Each of the 4 points (intervals) in this figure corresponds to one of the intervals in Figure 8. In this figure, 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 interval is not permitted to end before it has started, therefore no interval is allowed to reside in the lower-right part of this figure. Interpreting intervals as points in a 2-dimensional space was first done by Rit [1986], who described how qualitative Allen Interval constraints could be used and propagated on such representations[3] This representation was later discussed by Pujari et al. [2000] and

[3]The problem identified by Rit [1986] was named Sets of Possible Occurrences (SOPOS).

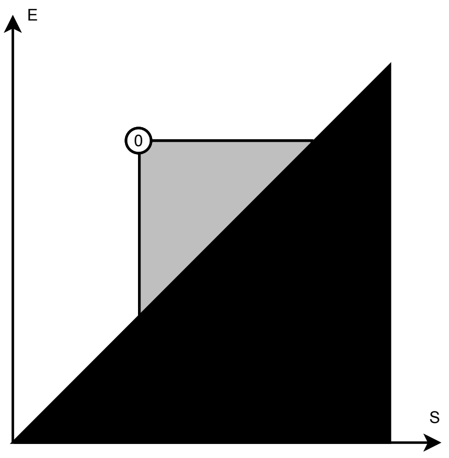


Figure 10: The set of intervals that occur `During` interval 0 in Figure 8.

has subsequently only been briefly mentioned in other work [Duftschmid et al., 2002, Aigner and Miksch, 2006]. The reason for this lack of attention is most likely because of the introduction of alternative problem formulations such as the STP, TCSP [Dechter et al., 1991] and DTP [Stergiou and Koubarakis, 1998]. However, in our problem the constraint networks are relatively simple compared to the ones in most contemporary work, whereas the sets of intervals that we want to reason about is large. Thus there is reason to believe that this representation is better suited for our particular problem.

Representing intervals as points in a two dimensional space not only serves as a visual aid, but more importantly, this representation can also be used to "generalize away" the usage of enumerated sets of intervals and instead consider groups of intervals. Figure 10 visualizes such a set of intervals. Specifically, the gray triangular area protruding from the diagonal in this figure corresponds to the set of all intervals that are `Contain`ed within Interval 0 in Figure 8. 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 9.

By looking at Figure 8 and Figure 9 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", during which we received no indication of this being true. Figure 11 illustrates the mentioned set. The rationale behind this might 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

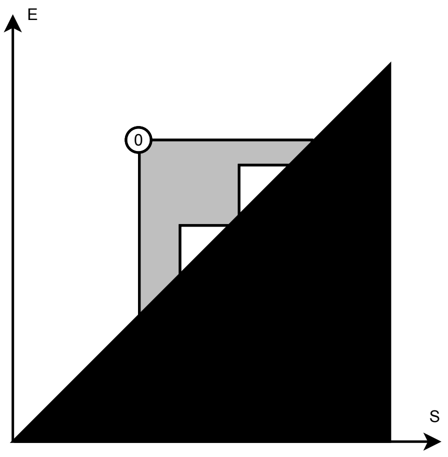


Figure 11: The set of intervals that occurs `During` interval 0 in Figure 9, excluding all intervals contained within the two "gaps" in the timeline.

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.

5.2.3 Constraints among multiple intervals

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 12. This figure shows one single interval i along with the set of intervals B that satisfies the temporal constraint i (`Starts` $\vee$ `Started-By`) $[l, u]$ b, so that any interval b in B starts at least l and at most u time units after i. 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 13, 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 12.

Furthermore, Figure 14 illustrates a constraint that involves more than a single timepoint 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

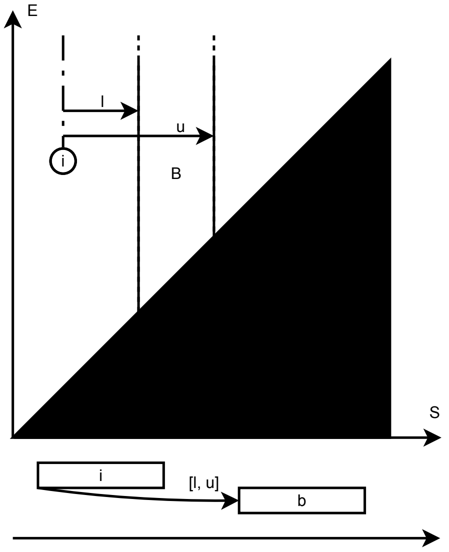


Figure 12: A ($\texttt{Starts} \vee \texttt{Started-By}$) $[l, u]$ constraint.

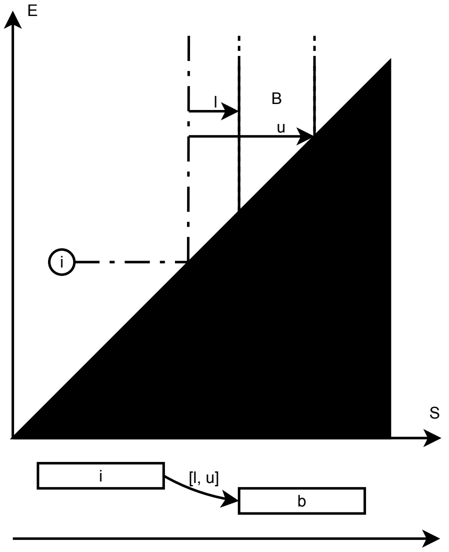


Figure 13: An $\texttt{After}[l, u]$ constraint.

is also limited to occur after the start time of i.

5.3 Algorithm

The propagation algorithm that is used to solve the context recognition problem is basically a reimplementation of the AC-3 algorithm [Mackworth, 1977], that is 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.

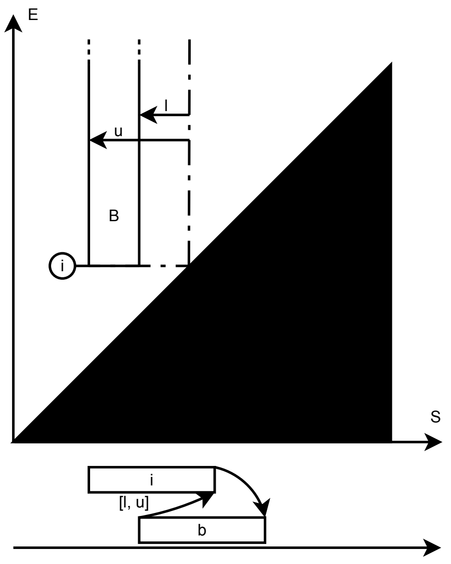


Figure 14: The `Overlaps`$[l, u]$ constraint.

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 Propagation algorithm

```
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* [de Berg et al., 1997] 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. This is done as shown in Algorithm 1.

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 in the algorithm 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 [1985]. Note that optimal decomposition of a simple polygon can be done in $O\left(r^2n^2\right)$ time [Keil, 1985], 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 $O\left(n\log(n)\right)$ time [Hertel and Mehlhorn, 1985] 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 timepoints). In the STP, these timepoints are initially constrained to each other with simple distance constraints reflecting the Allen interval constraint defined by c. Furthermore, one pair of timepoints 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 [Floyd, 1962], which reduces the temporal domain of the remaining pair of timepoints.

At this stage, the mutual temporal relationship between the second pair of timepoints 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 12–14. In these figures, the pair of timepoints 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 timepoints. Each such convolution of a single interval i generates four new intervals (i.e., the vertices of the admissible area of B in Figure 14). 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 15.

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

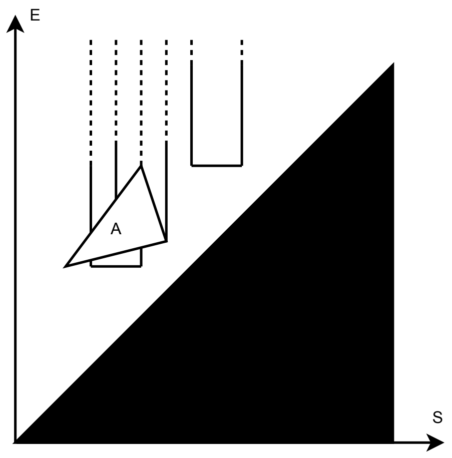


Figure 15: The individual convolutions of the intervals defined by region A with the constraints shown in Figure 14.

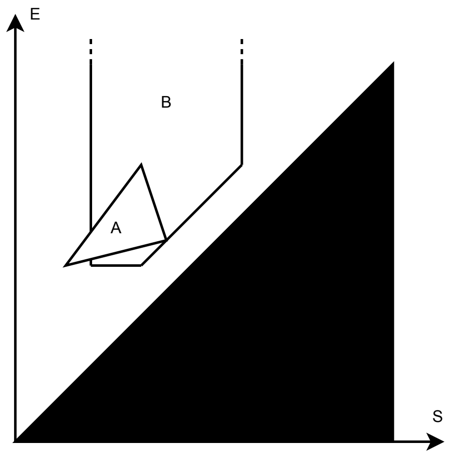


Figure 16: The convex hull of the individual convolutions shown in Figure 15.

c such as the one illustrated in Figure 16. The convex hull is retrieved with a Graham scan [Graham, 1972] which has a complexity of $O\left(n \log\left(n\right)\right)$ where n is the number of intervals (vertices) in I.

Finally, 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 [1998], which has a complexity of $O\left(mn\right)$, where m and n are the number of vertices (i.e., intervals) in two polygons. However, in practice, this can usually be done much faster if some sort of partitioning scheme is used.

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.

5.4 Example and experiment

In this section we present a concrete example of context recognition in an apartment which is based on data collected during an experiment. Although this experiment was done before the GiraffPlus first system integration, the sensors and the inference rule are very similar to those employed during the integration demo done at the end of 2012, corresponding to deliverable 5.2.

We also compare the performance and quality of the inference of the geometric approach against the results that can be obtained using the chronicle recognition system suggested by Dousson and Maigat [2007]. The data used for this evaluation was collected in the home of a researcher during a two-week long period with the help of a few wireless sensors. The sensors provide samples measuring movement, humidity, illumination and the usage of furniture such as the couch and the bed.

In the chronicle recognition approach originally described by Dousson and Maigat [2007], patterns of sensor events (chronicles) are recognized. In our approach, as well as other constraint based approaches to context recognition [Pecora et al., 2012], sensor readings and inferred activities are represented as intervals, i.e., pairs of timepoints. Nevertheless, the chronicle recognition technique can be easily extended to deal with intervals. Such an extension is what we are comparing against — which effectively makes chronicle recognition functionally identical to the approach described by Pecora et al. [2012].

The scenario used for this evaluation consisted of inferring an activity **WatchingTV** from two sensors whose data is taken to reflect if someone is **InLivingroom** and **InCouch**. Thus, the requirement for inferring **WatchingTV** is

$$\mathbf{WatchingTV}\ \texttt{Contains}\ \mathbf{InCouch} \land \mathbf{WatchingTV}\ \texttt{During}\ \mathbf{InLivingroom}.$$

Finally, any interval in the domain of the **InLivingroom** variable is required to have a duration of at least 5 minutes.

The data used to construct the **InCouch** timeline consisted of 120,581 samples from a pressure sensor mounted underneath a couch, and the **InLivingroom** timeline was constructed from 50,133 samples from a PIR sensor.

In the experiment we used a simple discretization method to generate intervals out of the raw data coming from a sensor. The discretization method used consisted of finding all intervals in which the mean of the samples is above a predefined threshold. Specifically, given a set of samples $s_0, \ldots, s_n$ for a sensor s, the samples' respective time $t_0, \ldots, t_n$ and a threshold T, we formed a set I_s containing all the intervals $[t_i, t_j]$ where

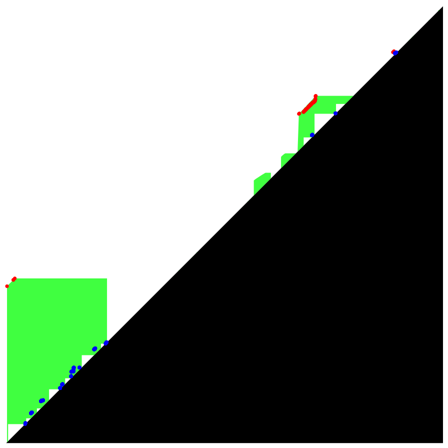


Figure 17: Comparison of the sets of intervals in the domain of **WatchingTV** that were recognized with the chronicle recognition system and the geometric approach. The green areas show the set of possible intervals obtained by the geometric approach and the red and blue dots show the earliest start-/latest end-time and earliest start-/earliest end-time solutions of the chronicle recognition approach.

$$\sum_{x=i}^{j} \frac{s_i\left(t_x - t_{x-1}\right)}{t_j - t_i} \geq T.$$

Furthermore, any interval that could be fully contained in another was discarded.

When applied to the samples, the discretization method generated 3,186 intervals for the **InLivingroom** variable and 4,764 intervals for the **InCouch** variable. In the case of the chronicle recognition, we inferred activities from these sets of intervals as they were. However, when inferring from these sets with the geometric approach, we used the convex hull of each cluster of overlapping intervals in I_s. The idea is that the convex hull provides us with a more general description of the data that does not contain unnecessary detail, a description that is difficult to obtain for an approach that uses enumerated sets of intervals as input. The sets created for the **InLivingroom** and **InCouch** variable consisted of 1,036 intervals in 175 polygons and 170 intervals in 26 polygons respectively.

During inference, the geometric approach took 2.6 s to finish propagating its network. In the chronicle recognition system, the corresponding operation took 154.4 s (average of 10 runs). After propagation, the domain **WatchingTV**, was defined by 1,086 intervals in 174 polygons. In contrast, the chronicle recognition system recognized the pattern 101,615 times (each one using a unique combination of intervals in the input data as support).

Clearly, the chronicle recognition system is not conceived to deal with input data representing alternative views of the sensor readings, and forcing it to do so obviously affects performance heavily. However, this comparison is intended to show the difference in the quality of the inference. Figure 17 shows the solutions to the context recognition problem obtained with the geometric approach and the chronicle recognition system. In this figure, the filled areas show the possible intervals in the domain of the **WatchingTV** variable inferred using the geometric approach. The dots that are farthest from the diagonal show 799 unique "earliest start time, latest end time" solutions obtained by the chronicle recognition system. I.e., since a solution to the STP contains flexibility in the timepoints, we have extracted the earliest possible start time and the latest possible end time of each interval recognized by the chronicle recognition system[4]. Similarly, 1,675 earliest start/end time solutions were extracted and are represented by the dots in the figure closest to the diagonal. Overall, the dots represent two sets of unique solutions obtainable with the chronicle recognition approach. We can further interpret the result of the chronicle recognition system to encompass a continuum of intervals between the two sets of dots. These intervals would be most likely supported by the input data. Note, though, that our approach, which can reason in terms of implicitly defined sets of intervals, achieves a result that is by far more informative. The geometric approach provides a result that is comparable to the chronicle recognition system's result, but is less brittle. Furthermore, note that there are two distinct sets of intervals (slightly to the right of the middle in the figure) that the geometric approach recognizes but the chronicle recognition fails to find.

As the propagated domain of the activity variable is represented in a two-dimensional space, one cannot directly use the domain of the activity variable to visualize the occurrences of events on a "traditional" timeline. This difficulty is, however, not exclusive to the geometrical approach. Note in fact that, since patterns matched with Allen interval constraints often contain residual flexibility, one timeline only offers a partial representation of the state of the world. The typical solution to this is to extract the earliest start-/ earliest end-time solution and take this to represent the precise occurrence of activities. This can of course be done also with the two-dimensional results of the our approach, however note that we would be discarding much more information: not only it is necessary to choose the bounds of intervals to represent on the timeline, but also the intervals themselves.

[4]Note that these solutions are however not guaranteed to be a valid in the general case.

5.5 Requirements

We have gone through the requirements for the Intelligent Monitoring and Adaptation Services according to **D1.3 System Reference Architecture** in order to ensire that the capabilities of the context recognition system matches those requirements. In the following we present each requirement and indicates if and how it could be fullfilled. Our conclusion is that the capabilities of the context recognition system under development should be adequate.

Serial/Ref and Descriptor	Requirement Statement	Context inference
1.a.7 Determining whether the person suffers from incontinence	GiraffPlus shall help in understanding whether the elderly person suffers from incontinence over different day periods for a variable amount of time	Incontinence sensor
1.a.10 Monitoring sleep activity	GiraffPlus shall monitor sleep activities over different day periods for a variable amount of time	Bed sensor + motion sensor
1.c.1 Detecting the position	GiraffPlus shall detect the position of the elderly person inside the house over different day periods for a variable amount of time	Activities for each position, based on motion sensors
1.c.2 Monitoring the movement	GiraffPlus shall monitor the movement inside the house over different day periods for a variable amount of time	Motion sensors
1.c.3 Detecting the absence of movement	GiraffPlus shall detect the absence of movement of inside the house over different day periods for a variable amount of time	Motion sensors, take the complement (i.e. no movement in a room by movement sensor giving 0, for entire home take conjunction of those).
1.c.4 Temporal monitoring of the position	GiraffPlus shall temporally monitor the elderly person's position inside the house	Same as 1.c.1 Detecting the position

Serial/Ref and Descriptor	Requirement Statement	Context inference
1.c.6 Monitoring of night activities	GiraffPlus shall monitor the night activities of the elderly person inside the house over different day periods for a variable amount of time	Multiple activities including 1.c.4, 1.c.14, 1.c.16
1.c.7 Monitoring cooking ability	GiraffPlus shall monitor the cooking activities of the elderly person inside the house for a variable amount of time	Electrical socket sensor of stove, microwave, door sensor of fridge, motion in kitchen
1.c.8 Monitoring the time spent for preparing lunch	GiraffPlus shall monitor the time spent by the elderly person for preparing lunch for a variable amount of time	Fridge, motion in kitchen, with time constraints
1.c.9 Monitoring the time spent in different home places	GiraffPlus shall monitor the time spent by the elderly person in different places inside the home for a variable amount of time	Same as 1.c.1 Detecting the position
1.c.10 Monitoring the use of refrigerator at home	GiraffPlus shall monitor the frequency with which the refrigerator is opened by the elderly person over different day periods for a variable amount of time	Fridge door sensor
1.c.11 Monitoring the social interactions activity	GiraffPlus shall monitor the frequency of the social interactions of the elderly person over different day periods for a variable amount of time	This particular requirement demands more elaboration if it should be recognized as an activity.

Serial/Ref and Descriptor	Requirement Statement	Context inference
1.c.12 Detecting decline in mobility	GiraffPlus shall help monitoring a decline in the mobility of the elderly person for a variable amount of time	Statistics of 1.c.2 Monitoring the movement
1.c.13 Detecting absence of the elderly person	GiraffPlus shall detect the absence of the elderly person in the house during unusual period for a variable amount of time	Based on 1.c.3 Detecting the absence of motion with time constaints, and/or on main door opening, and/or GPS
1.c.14 Monitoring the use of home appliances	GiraffPlus shall monitor the use of home appliances by the elderly person over different day periods for a variable amount of time	Socket sensor and/or appliance door sensor
1.c.15 Detecting Falls	GiraffPlus shall detect whether the elderly person falls inside the house for a variable amount of time	Fall sensor (or absence of motion with time constraints.)
1.c.16 Monitoring the time spent in bed	GiraffPlus shall be able to monitor how much time the elderly person spends in bed over different day periods for a variable amount of time	Bed occupancy sensor
1.c.17 Monitoring person balance	GiraffPlus shall help monitoring the elderly person's ability to maintain balance for a variable amount of time	Possibly by balance sensor under development

6 Configuration Planner

The purpose of the configuration planner in the GiraffPlus project is to provide the context recognition with the data it needs for recognizing the activities requested by the (mainly secondary) users. The data comes from observable state variables. The inclusion of the configuration planner allows us to separate the descriptions of sensors and preprocessing of sensor data from the more abstract descriptions of activities, which in turn allows the GiraffPlus system to use the same activity descriptions and perform the same inferences in different apartments with different sets of sensors available, or in the same apartment as sensors are added, removed, or fail. In addition, the GiraffPlus system can *automatically* adapt to varying sensors, without the need for user intervention. Likewise, the configuration planner helps to separate the description of actuation devices from the more abstract representation of enactment of the context recognition (although actuation is less frequent than sensing in the context of the GiraffPlus project).

A central concept in configuration planning is a configuration, which informally is a set of selected sensors, actuators and computational processes that exchange information in order to solve a task. This task may be to observe one or more state variables, and/or to change some state variables. Some task may also require several steps, in particular if actuation is involved.

The configuration planner is capable of finding configurations that contain consistent causal and information links between sensors, actuators and programs. In this way, given a world representation and a goal, a configuration planner can deliver fully admissible configurations that satisfy the goal. The world representation contains a finite list of available functionality instances, and a set of state variable assignments in its starting state. The goal should be specified in terms of state variables (to observe), or state variable value assignments (to change). Functionality instances represent sensors, programs and potentially actuators.

The approach we present here is inspired by the work of Lundh et al. [2008]. However, whereas that earlier approach used a combination of two different planners to handle causal and information dependencies, the approach proposed here handle both these aspects in a single algorithm. In addition, this approach is intended to facilitate easy monitoring and repair of configurations, as well as easy incorporation of new goals into an existing configuration.

6.1 Representation

6.1.1 State Variables and States

For referring to state variables, we use a minimalist language with the following format:

$$< \text{entity} > . < \text{property} > . < \text{format} >$$

A state variable assignment is written in the following manner:

$$< \text{entity} > . < \text{property} > . < \text{format} >=< \text{entity} >$$

For instance:

livingroom.temperature.celcius

bedroom.temperature.fahrenheit = 70

Also static information such as the location of sensors can be encoded as state variables:

sensor0.location.rooms = kitchen
sensor0.location.position = microwavedoor
microwavedoor.location.rooms = kitchen
sensor1.location.rooms = balcony
sensor1.location.position = balconydoor
balconydoor.location.rooms = balcony

A complete assignment of values to state variables is called a state, and states may change over time.

6.1.2 Functionalities

Functionalities are programs, possibly embedded in sensor and/or actuation hardware, that when a condition is met, and when provided certain information as input, can produce another information output and/or have a certain effect.

In GiraffPlus, most functionalities correspond to sensors and programs that aggregate sensor information to allow different levels of abstraction. The functionalities can be associated with conditions such as permissions from the users to run a particular monitoring tool, conditions related to the interpretation of a sensor reading, or just what is required to run a program. As an example, in order to consider a pressure sensor as both a bed occupancy sensor and a (partial) bedroom occupancy sensor, then that sensor must be in the bed, and the bed must be in the bedroom.

A functionality has the general form:

name:	< string >
input:	< state variables >
output:	< state variables >
precond:	< logical combination of state variable assignments >
effects:	< conjunction of state variable assignments >

For instance, consider a functionality that is a thermostat that sets the temperature of the livingroom to 70 degrees, and that needs the current temperature of the livingroom (in Fahrenheit). A simple way to describe such a functionality can be:

name:	thermostat_livingroom
input:	livingroom.temperature.fahrenheit
output:	–
precond:	–
effects:	livingroom.temperature.fahrenheit = 70

Note how the input is a state variable that will be observed, whereas the effect is a state variable assignment, representing a change in the world state.

Or consider again the bed pressure sensor, which can be used as a partial bedroom occupancy sensor:

name:	partial_occupancy_bedroom
input:	–
output:	bedroom.occupied–partial.bool
precond:	sensor0.bedoccupancy.bool = 1 *and* sensor0.location.room = bedroom
effects:	–

A functionality schema is a parameterized representation of a set of functionalities. In the following example, the parameters are *room* and *temp*, and by setting them to livingroom and 70, the functionality thermostat_70 above is obtained (but with the name thermostat(livingroom,70)):

name:	thermostat
param:	*?room*, *?temp*
input:	*?room*.temperature.fahrenheit
output:	–
precond:	–
effects:	*?room*.temperature.fahrenheit = *?temp*

6.1.3 Satisfying causal requirements (preconditions)

The causal requirements of a functionality instance are specified in the preconditions (**precond**). In a functionality instance, the causal requirements of the functionality need to hold when the instance starts being executed. The effects of the functionality instance are expected to hold after the execution is completed. Hence, a precondition of a functionality can only be satisfied by either the effect of a different functionality instance, or by the initial state. The initial state is the assignment of values for the state variables before any functionalities have been executed.

To satisfy a causal requirement with the effect of another functionality, the effect should operate over the same entity, property and units of the causal requirement. In such a situation, there is a causal link between the two functionalities.

For example, given the precondition:

livingroom.temperature.fahrenheit = 70

a functionality instance with an effect of changing the temperature of the livingroom to 70 degrees Fahrenheit is needed. (It is advisable not to mix formats for the same entity-property pair in the preconditions and effects of functionalities.)

Note that the state may have changed after one or more functionalities have been executed. This can lead to unintended interferences between functionalities, for instance if one functionality negates a precondition of a later functionality. Care must be taken to avoid such interferences. In this way, causal link are different from information links, which cannot interfere with other information links as they don't share any state.

6.1.4 Satisfying information requirements (information inputs)

Information requirements means that each input of a functionality must be connected to a matching output of another functionality. This represents a flow of data from the output of the latter functionality to the input of the former functionality. For instance, the functionality thermostat(livingroom,70) cannot work by itself. It needs to have some other functionality instance providing data for the state variable:

livingroom.temperature.fahrenheit

That information can be satisfiable by a functionality instance that has a matching output, such as:

livingroom.temperature.fahrenheit

which can for example come from a functiality instance thermometer(livingroom), instantied from:

name:	thermometer
param:	*?room*
input:	
output:	*?room*.temperature.fahrenheit
precond:	–
effects:	–

By linking the output of thermometer to the corresponding input of thermostat, the information requirements of thermostat can be satisfied. Note that this requires the two functionality instances to execute simultaneously. The link itself can be performed as e.g. a subscription from the functionality with the input to the functionality with the output.

6.1.5 Configurations

A configuration is a set of connections between a subset F of all available functionality instances in the world $\boldsymbol{F}$, in such a way that a goal is satisfied. In this work, we will use the definition of configuration in Equation 1.

$$c = \langle F, TC, IL, CL \rangle \tag{1}$$

Here, F is the set of functionality instances used in the configuration, and TC is a set of time constraints for each functionality: whether it is before, after, or at the same time as other functionality instances. IL is the set of information links, and CL is the set of causal links. Note that not all functionalities need to be executed at the same time.

A causal link between two functionality instances implies a sequential execution constraint between these functionalities, in which the functionality instance requiring an effect comes after the functionality that causes the effect.

An information link implies simultaneous execution of the two functionalities.

For a configuration to be fully admissible, all causal and information requirements should be satisfied in a way that no time constraints are violated, and in a way that no effect of any functionality instance violates a causal link.

To satisfy information admissibility (admissibility on information requirements) in a configuration, all inputs in each functionality instance should be linked to the output of another functionality. To satisfy causal admissibility (admissibility on causal requirements), all preconditions in a functionality instance should be satisfied before its execution. The set of links should be acyclic. When both information and causal admissibility are satisfied in a configuration, and no conflicts are present, we can say that such configuration is fully admissible, and consider it a candidate configuration for satisfying the goal.

6.1.6 Configuration planning problem

A configuration planning problem consists of a goal, an initial state and a set of functionality instances.

The goal is given in the form of a set of information and/or causal requirements that need to be satisfied. The planner then converts these elements into a partial configuration that contains a stub functionality instance with the same causal and information requirements as the goal contains. By representing the goal as a functionality, it is also possible to use the same algorithm to take goals that are partial configurations that need to be fully satisfied. It also makes the planner able to repair failed configurations from their point of failure. The initial state is also transformed into a functionality, with the state variables of the initial state as effects.

6.2 Algorithm

Algorithm refalg for configuration planning searches in the space of partial configurations. It is based on flaw repairing. In each expansion, a child represents the repair of a flaw in the parent configuration. There are two types of flaws: missing inputs, and missing preconditions.

For resolution mechanisms in conflicts between effects and existing causal links, one can either:

1. Move the execution time of the threatening functionality to a time strictly before the functionality instance that is the origin of the threatened causal link (promotion),

2. Move the execution time of the functionality instance with the threatening condition to be at a time later or equal to the time of the functionality instance that receives the threatened causal link (demotion),

The same algorithm can be adapted to adding new goals to an existing configuration. In such a case, the input includes the existing configuration including links and time constraints in addition to the initial state functionality f_i and the goal functionality f_g. The new goals are added to f_g, and will constitute flaws when the algorithm starts.

This algorithm also makes it possible to remove a failing functionality instance from the current configuration (as well as any other functionality that supports it by causal or information links, and are not needed for anything else). The removal of f and causal and information links that f supported will then constitute flaws in the remaining configuration plan, and the algorithm can attempt to repair them.

One of the challenges of the configuration planning problem, is that the search may have a high branching factor. In order to reduce the branching factor, discarding no-good configurations early, and obtain the configurations with the higher chances to be repaired in case of a failure, our planner presently guides the search with a histogram-based heuristic chain. Other kinds of heuristics will also be investigated during year two and three.

The heuristic chain tries to first satisfy the partial configurations with the higher chances to be satisfied, starting with the most failure-prone unsatisfied requirement in that functionality. The most failure prone requirement is the one that has the least amount of support in the histogram of outputs and effects i.e. in the case of a causal requirement, the precondition with the smaller number of effects that can satisfy it from the available functionalities, and in the case of an information requirement, the input with the smaller number of information outputs that can satisfy it. The partial configuration with the higher chances to be satisfied depends on the heuristic score of the configuration given a certain function. Our current function provides a score for a configuration, in which if

Algorithm 2 Configuration planner

Input: a set of functionality instances $\mathcal{F}$, an initial state functionality f_i and a goal functionality f_g.
Output: a configuration C.

```
1. Let $C = \langle \emptyset, \emptyset, \emptyset \rangle$.

2. If there are no more flaws, return the current configuration $C$

3. Select a flaw $g$ of some functionality instance $f$ (backtrack point)

4. If $g$ is a precondition flaw of $f$:

   (a) Select an existing functionality instance $f'$, or add to $C$ a new one $f'$
       from those available in $\mathcal{F}$, with $g$ as effect (backtrack point)
   (b) Add a causal link from $f'$ to $f$ labeled with $g$ to $C$.
   (c) Constrain $f'$ to come before $f$ in $C$ and, if $f'$ is new, to come after $f_i$
       and before $f_g$, and check temporal consistency
   (d) If the new causal link (from step b) is threatened by an effect of any
       other functionality instance, use resolution mechanisms to remedy this
       and check temporal consistency (backtrack point). Repeat until no
       such threat remains.
   (e) If $f'$ is new and has any effect that threatens any other causal link, use
       resolution mechanisms to remedy this and check temporal consistency
       (backtrack point). Repeat until no such threat remains.

5. If $g$ is an input flaw of $f$:

   (a) Select an existing functionality instance $f'$, or add a new one $f'$ to $C$,
       with $g$ as output (backtrack point)
   (b) Add an information link to $C$ from $f'$ to $f$ labeled with $g$.
   (c) If $f'$ is new and has any effect that threatens any existing causal link,
       use resolution mechanisms to remedy this and check temporal consis-
       tency (backtrack point). Repeat until no such threat remains.
   (d) Constrain $f'$ and $f$ to be at the same time in $C$ and check temporal
       consistency.

6. Goto 2.
```

an unsatisfied requirement has a zero frequency in the histogram, the score of that configuration is exactly zero, and the partial configuration can be discarded from the search stack. The number of ways in which each unsatisfied requirement can be satisfied is aggregated and normalized by the number of unsatisfied requirements. Since our configuration planner is expected to be able to handle goal preferences in the next version, we allow this function to depend on the type of performance expected for the functionality. For instance, if the goal preference is the configuration with the fastest execution times, the number of causal links should be minimized, and configurations containing functionalities with the smallest execution time can be favored. Similar functions can be applied for any other particular preference.

The term heuristic chain is used to the act of combining a heuristic for choosing the next node to satisfy, and another one to choose the next link to satisfy. It is similar to combining variable ordering (choosing the next node to satisfy) and value ordering (choosing the next unsatisfied link to satisfy) in constraint satisfaction problems. Using histograms as a heuristic for value ordering is natural in the context of our problem [Liu and Havens, 2004], and especially for obtaining configurations that can easily be repaired, because the availability of "spare parts" is considered while planning. Using a function based on histograms for selecting the next configuration to satisfy allows this trait to be even more emphasized while planning. Combinations of heuristics can also be performed, in which several heuristics are used for pruning, and one for ordering the pruned search stack. Also, combinations of heuristics have successfully been applied for speeding up planning. For example, in the FastDownward planning system [Helmert, 2006], having several options for combining different search algorithms and also different heuristics is one of the features that allow the algorithm to succeed. There is ongoing work to make the configuration planner combine heuristics not only for pruning, but for using different heuristics as evaluation criteria.

6.3 Configuration planning example

The following is a short example intended to illustrate how the configuration planner works. It is not selected from the requirements, but it has been chosen because it illustrates both causal and information (sub)goals in a simple way. The caregiver and the user decided that in order to have a better sleep, the user needs a higher bedroom temperature. A request is sent to the planner to satisfy the goal `bedroom.temperature.fahrenheit = 75`. Among the functionalities that we have in the world, there are:

```
Thermometer
(gives the temperature of a room with a therometer in Celsius)
Param: ?room
Precond: ?room.has-thermometer.bool = True
```

```
Outputs: ?room.temperature.celsius

Thermostat
(changes the temperature of a room to a value,
both provided as parameters,  operates over
Fahrenheits. The range of temperatures goes
between 68 and 86 Fahrenheit)
Param: ?temp={integers between 68-86},
            ?room={rooms taken from the rooms defined as state variables}
Input: ?room.temperature.fahrenheit
Precond: ?room.has-thermostat.bool = True
Effect: ?room.temperature.fahrenheit = ?temp

C-F-Converter
(converts the temperature of something from Celsius to Fahrenheit)
Param: ?entity={*}
Input: ?entity.temperature.celsius
Effect: ?entity.temperature.fahrenheit
```

There are also the following initial and goal functionalities, generated from the current state and the requested goal:

```
Initial
Effect: bedroom.has-thermometer.bool = True,
        bedroom.has-thermostat.bool=True,
        ...

Goal
Precond: bedroom.temperature.fahrenheit = 75
```

Instances of these functionalities constitute the initial configuration which is given as input to the planner.

The precondition of `Goal` constitutes a causal flaw. It can be resolved by adding a functionality instance `Thermostat` with `?room=bedroom` and `?temp =  75`. A causal link labeled `bedroom.temperature.fahrenheit = 75` from this functionality to `Goal` is also added, and temporal consistency is check (this is done each time a temporal constraint is added).[5]

Now there are two new flaws: `bedroom.has-thermostat.bool=1` (causal) and `bedroom.temperature.fahrenheit` (information). The first of the flaws can be resolved with a causal link to `Init`. The second one can be resolved by adding

[5]If there had been more ways to control the temperature in the bedroom, the planner would have created a partial configuration for each such way, and addressed them in an order determined by the histogram-based heuristic. However, in this particular example we only consider one way to resolve each flaw.)

a functionality instance `Converter` with `?entity = bedroom` and an information link from this instance to the `Thermostat` instance. In GiraffPlus, this link would imply that the latter instance would subscribe to a topic of the former. Temporally, it implies that the two instances must run at the same time.

However, the new instance requires an input `bedroom.temperature.celsius`, and this becomes a new flaw in the partial configuration. The new flaw can be resolved by adding an instance `Thermometer` with `?room = bedroom`, and adding an information link from the output of this new instance to the input of the `Thermostat` instance.

The two flaws from the `Thermostat` functionality instance have been eliminated, but a new one appears: `bedroom.has-thermometer.bool = True`. Fortunately, this flaw is easy resolved by a causal link from the `Init` functionality.

At this stage there are no flaws remaining, and the current configuration is admissible as well as temporally consistent. The planner returns this configuration.

6.4 Configuration Planning vs Partial-Ordered Task Planning

Our approach is similar to partial-ordered planning (more particularly POP [Weld, 1994]), but with important differences that emerge from the interaction of information and causal requirements. In this section, we will explain the differences.

POP is a regression planner that operates on the space of partially-ordered plans. A partially-ordered plan consists of a set of steps, a set of ordering constraints between the steps, and a set of causal links also between the steps. Each plan step is a STRIPS operator, with instantiated variables. A causal link $S_i \xrightarrow{c} S_j$ state that S_i achieves precondition c for S_j. Planning in POP starts with an initial plan that should be refined until a solution plan is obtained. The initial plan contains a start state and a goal state as its only two steps. The start step has no preconditions, and has the initial state of the world as its effects. The finish step contains no effects, and has the goals as preconditions.

In our configuration planner, a configuration contains a set of functionality instances, ordering constraints between the functionality instances, a set of causal links and a set of information links between the functionality instances. The causal links are treated in the same way as in POP (i.e. threats are resolved by promotion or demotion). However, functionality instances differ from steps (STRIPS operators) in the fact that they can have information inputs and outputs. The presence of an information link between an output and an input does not explicitly imply a change in the world state: sending information doesn't erase other information. Therefore, there cannot be conflicts between information links. What an information link does imply, is an execution time constraint of simultaneousness between the functionality instances that generate the input and the output involved on the link.

In our configuration planner, since a functionality instance could at the same time satisfy an information requirement for one instance, and a causal requirement for another instance, the mechanisms for finding conflicts have to consider not only sequential but also simultaneous execution. Fully admissible configuration plans, just as POP solution plans, may still remain partially-ordered, allowing configuration plans to be as flexible as possible. When a configuration plan is chosen for execution, a scheduler can choose to assign execution times that are consistent with the constraints between the functionality instances.

6.4.1 Properties

POP is complete and optimal as long as its search strategy is complete and optimal (note that POP is not restricted to any particular search strategy). The same can be applied to this approach of configuration planning, since it shares the same features of POP that allow this statement to hold. However, it has the same drawbacks: the branching factor can be high, especially considering conflict resolution. The use of the heuristic chain for search stack ordering and pruning is intended to reduce branching factor and to achieve optimality. However, as information links cannot by themselves lead to conflicts (only causal links can do that) we consider the branching factor a lesser problem in configuration planning than in partial-order task planning. As a matter of fact, the information links can be efficiently managed by the kind of regressive search performed by our configuration planner.

6.5 Implementation

We have implemented our own configuration planning framework in Java. This framework is intended to facilitate testing and comparisons of different configuration planning approaches. It contains two basic packages:

- One package that contains classes for functionalities, states, and configuration plans.

- One package that contains classes for performing search. This package supports implementing and testing alternative search strategies.

6.6 Ongoing work

The next version of our configuration planner will be able to plan with partially-ordered preferences. Preferences are desirable because they can support resource handling (whether using the expensive test stripes or not, depending on whether we prefer a quick test or if we can wait until the stripes are fetched), they can allow the planner to get more “preferable” configurations , and because they can also be used to express richer goals for the whole system (a functionality can express its requirements in terms of preferences, e.g. a service could prefer to use the information gathered by the pressure sensor in the bed, than the information gathered by the movement sensor in the bedroom, in order to determine the amount of sleep that a person had).

The mathematical framework that we are using to represent partially-ordered preferences is the semiring-based soft constraints framework Bistarelli et al. [1999]. This will have a significant effect on the definitions of a functionality, a configuration, causal and information links, and admissibility.

7 Integration of Configuration Planner and Context Recognition

The interaction of configuration planning and context recognition is performed with Java remote method invocation (RMI). RMI allows the context recognition to call methods and retrieve data from the configuration planner.

The most basic form of interaction is that the context recognition requests a state variable, such as `couch.occupied.boolean`, from the configuration planner. The latter responds directly with the corresponding sensor name, which can be uses to access data from the sensor in the data storage.

During a transition period there is the need to still allow sensors to be referenced directly, i.e. by using the database name of sensors in the configuration file that describes the rules that are used by the configuration file. In order to facilitate this transition goals to the configuration planner are prepended with an `@` character, while direct sensor references are written without the prepended character.

The following preprocessing statement contains a request for the configuration planner to resolve the database name for a couch occupied variable:

```
<preproc name="average" in="@couch.occupied.boolean" out="in_couch"
lim="0.99"/>
```

The same direct reference looks like this:

```
<preproc name="average" in="sensors.livingroom_couch.couch_occupied"
out="in_couch" lim="0.99"/>
```

In general, when the context recognition needs to monitor some activity, a set of information goals is generated by selecting the state variables to monitor, according to the inference rules considered in each case. These goals are transfered to the configuraion planner by RMIs.

The configuration planner should choose a configuration after it finishes planning. The configuration planner currently guides the search with a histogram-based heuristic chain, in which it tries to satisfy the configurations with the higher chances to be satisfied, starting with the most failure-prone unsatisfied link in that functionality. This heuristic leads to choosing the plans with the higher chances to be repaired in case of a failure, given the current conditions of the world representation. The first configuration found is the safest configuration, in which repairs are easier to perform. Unless anything is specified as a goal preference (ongoing work), the safest configuration are chosen by default.

After the configuration is chosen, the configuration planner should execute it. Execution of a configuration implies setting subscriptions of functionality inputs to the proper functionality outputs, allowing the execution of a functionality to satisfy a causal requirement, and combining these two actions with the execution constraints of the configuration. After all subscriptions are set for information links, and all causal requirements are sequentially satisfied for causal links, the outputs of information goals are directed to the output topic of each goal. Finally, the conguration planner returns the database key for accessing the resulting data. If the data comes directly from a sensor, then the sensor name is returned. Otherwise, the topic for the output of the relevant functionality is returned.

If no acceptable configuration was found, the configuration planner returns a message "error: no configuration was found" message on the output topic of the failed goal.

We expect this protocol to develop further as we continue to develop the interaction between context recognition and configuration planning. Therefore, the protocol has been designed to be easily extendible.

8 Deployment

In this section we describe the flow of data from the sensors, through the context recognition, and finally to the database. We also describe how the context recognition module support the context recognition in accessing sensor data.

Currently the working context recognition system, including the custom made sensors utilizes both the MQTT middleware (see mqtt.org) and a REST API (HTTP) to retrieve sensor data from the database. In addition, the ORU-specific PEIS middleware middleware (similar to MQTT) is currently used for internal communication in the CR system but will be phased out.

The sensors currently used for context recognition in the test apartment in Ängen are the custom ones provided by ORU and the ones provided by Tunstall. The data from the custom sensors are recieved by a USB-“basestation” running on a Linux machine in the apartment. This data is then re-interpreted at different levels and finally published on MQTT. In this chain the data consists of “unit-less” AD-readings, which are eventually transformed into a more meaningful representation, depending the type of sensor and by applying different thresholds. For instance, temperature can be represented in degrees Celsius, or occupancy for determining when someone is sitting on a couch can be specified as a weight threshold for a pressure sensor.

The Tunstall sensors only provide boolean readings, such as motion from a pir sensor, stats of an electical appliance, or bed occupancy. Thus, these sensors provide similar functionality in most cases, but their configuration are mostly done physically on the sensor itself. These sensors are connected to a gateway running on a windows computer in the apartment and publish their data directly into MQTT in the same topic format as the one utilized by the custom sensors, an example of the data format of is included in Figure 18.

As mentioned, the MQTT middleware works like a distributed database, relaying data to all connected listeners. This means that it is possible to connect to MQTT to get a current view of the ongoing events in the apartment, i.e. by being notified of a change rather than having to query for the state. However, the most important listener is the database listener running on the database server. This listener stores the published data persistently in the database and makes it available to other components over the REST protocol.

The database is accessed by the context recognition system which infers activities form the data contained therein. This can be done in two ways. The context recognition can run in an off-line mode, and in this mode a particular window of time is provided to the CR system. The data contained in the window of time is then fetched from the database over the REST protocol and is preprocessed to generate the higher level descriptions of sensory events. In the other case, the context recognition system is supplied with a start date, from which it then continuously downloads all following samples. This is currently done approximately each third second, but this frequency can be adjusted accordingly to how crucial

```
{
    "home" : "angen",
    "room" : "bathroom",

    "category" : "xbee",
    "interval" : "3",
    "onchange" : "no",
    "type" : "ir",
    "unit" : "boolean",
    "sensor_id" : "bathroom.motion",

    "time" : "2013.01.15.13.23.24.758",
    "value" : "false"

    "last_time" : "2013.01.15.13.23.21.831",
    "last_value" : "false",
}
```

Figure 18: Example of a JSON entry describing one received sample.

it is to detect an activity on time. Regardless of the polling frequency, the final result should be the same. Furthermore, downloaded data is stored locally and encrypted on the computer running the context recognition in order to minimize retransmissions of the same data. For privacy reasons the encryption key can be set to the same as the key used to access past data from the database, thus this imposes no overhead in terms of security.

As mentioned, the downloaded data is used to form higher level descriptions of the sensory events in the apartment by the preprocessing component. This can include, for instance, generating a set of intervals that define when a person is lying in bed from the discrete samples coming from a bed pressure sensor. These sets of intervals are maintained locally and are available to the actual context recognition engine which reads these, applies the temporal constraints accordingly to the domain description and generates new higher level defining activities which are grounded in the sets of preprocessed intervals.

Finally, a timeline extraction component is continuously monitoring the set of activities for changes. When a change is detected one descriptive timeline is extracted from the multiple possibilities defined by the sets of intervals and published back into MQTT. Due to the fact that the extracted timelines does not have a “natural identity” (i.e. it is difficult to associate one instance of a performed activity with another one extracted a few seconds after in the presence of additional data) the interval data sent over MQTT invalidates past contents of the database for a defined period of time. An example of a JSON entry sent over MQTT is included in Figure 19

```
    "activity_name" : "act_in_bed",
    "update_start" : "2012.12.19.14.00.00.000",
    "update_end" :   "2012.12.19.15.00.00.000".
    "home" : "angen",
    "room": "bedroom",

    "intervals" : [
        {
            "start" : "2012.12.19.14.16.29.000"
            "end" :   "2012.12.19.14.47.45.000",
        },

        {
            "start" : "2012.12.19.14.52.32.000"
            "end" :   "2012.12.19.15.00.00.000",
        }

]
```

Figure 19: Example of a JSON entry describing a set of extracted intervals.

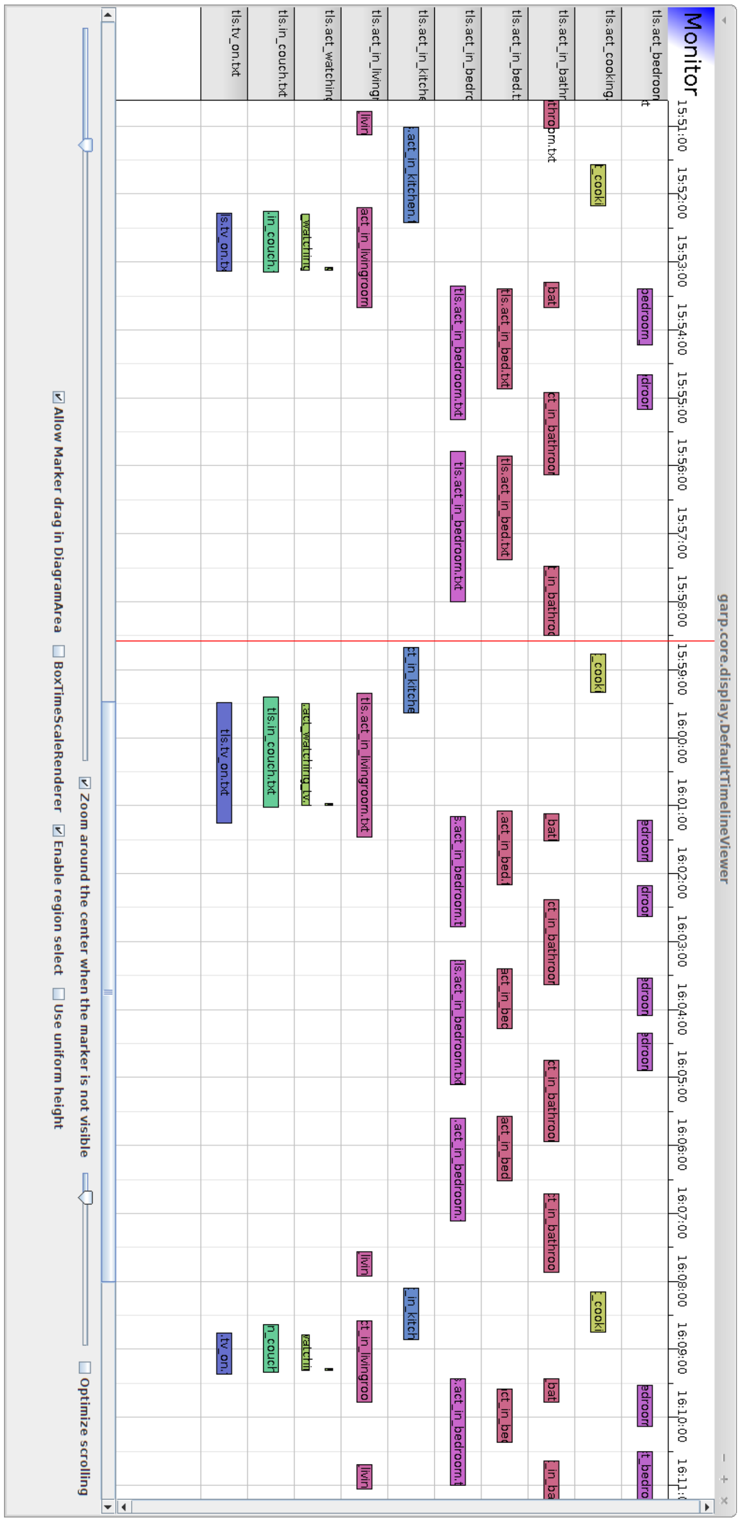
As previously mentioned, the configuration planner allows us to separate the descriptions of sensors and preprocessing of sensor data from the more abstract descriptions of activities, which in turn allows the GiraffPlus system to use the same activity descriptions and perform the same inferences in different apartments with different sets of sensors available, or in the same apartment as sensors are added, removed, or fail. In the current version of the GiraffPlus system, the main role of the configuration planner is to match the state variables requested by the context recognition to the available sensors, as is described in section 7. In addition, the configuration planner has the capability to add functionalities that perform processing of sensor data such as transformations and thresholds. These functionalites are presently implemented as Python scripts and utilize the MQTT middleware for accessing data from the sensors and storing in the data storage. They are invoked from the configuration planner as Linux processes.

9 Demonstration

In order to assess the functionality of the current versions of the context recognition and configuration planning systems we exposed them to a series of tests performed in Ängen, which is an apartment at a residence for elderly in Örebro which has been used as a first test site. During these tests we performed activities in the monitored environment that were thought to be typical for elderly people, and which were included in deliverable D1.1 User Requirements & Design Principles Report. Data from these test are also included in the video demonstration (Deliverable 5.2 Video-milestone 1). In short, the tests consisted of a subject moving around in the apartment in order to verify that the localization of the inhabitant is working correctly (e.g. that the passive infrared sensors were working as intended). Additionally, we monitored **Sleeping**, **Showering**, **Cooking** and **WatchingTV** activities. The demo consisted of performing a 7 day scenario in the apartment following a pre-written script. Due to practical limitations the 7 day script was performed at a more rapid pace (e.g. instead of lying in bed for 9 hours sleeping the same activity was reduced to a few minutes). Figure 20 shows parts of the timelines generated during the demonstration.

The sensors that were employed in this demo were the custom sensors made by ORU and the Tunstall sensors. The data from these sensors were continuously sent to and stored in a database at XLab. The storage system had been tried extensively prior to this demo, at the time of writing the database contained approximately 8 million samples collected from the Ängen apartment. The work of setting up the network and integrating the sensors were part of WP2 and WP6.

During the demo the context recognition and configuration planning systems were started on a computer in the Ängen apartment, and although the recognition was performed at the site this was purely a matter of convenience at the time. Since the context recognition's sole source of samples is the database there are no physical limitations on where the software is actually run, in fact, each module of the context recognition system can in principle be run at different locations.



[✓] Zoom around the center when the marker is not visible
[ ] Optimize scrolling
[✓] Allow Marker drag in DiagramArea
[ ] BoxTimeScaleRenderer
[✓] Enable region select
[ ] Use uniform height

Figure 20: Part of timelines generated during demonstration)

10 Conclusions

In this report, which constitutes deliverable 3.1 of WP3 about Intelligent Monitoring and Adaptation Services, we have presented prototypes of the context recognition and configuration planning components. These prototypes have been used in the M12 demonstration of WP5.

A prototype of the context recognition module has been developed in **task 3.1 Long-term context recognition**, and we asses that its capabilities matches the requirements. During the first year, focus has been on the context inference level, and in particular to make inference more robust in case of missing data. During years two and three, we in addition intend to develop methods for data interpretation in order to go from continuous signals to abstract state variables, to merge data from several (redundant) sources including physiological sensors, to handle context-dependency (e.g. time of the day) in data interpretation and inferences, and to handle uncertainty.

A prototype of the configuration planner has been developed in **task 3.2 Configuration planning for efficient activity recognition**. An approach inspired by partial-order planning has been used, as we consider this suitable for information goals and it facilitates adding goals incrementally as well as re-planning in case a device malfunctions. During years two and three, we intend to explore efficient heuristics for the planner, and we intend to include preferences in order to handle conflicting objectives. In addition, we will explore the dynamics of configuration planning, including arrival of new goals and completion of old goals, and monitoring and repair of configurations.

An early integration of context recognition and configuration planning has been developed in **task 3.3 Integration of context inference and configuration planning**. This integration consists of a protocol which allows the context recognition component to request goals in terms of state variables from the configuration planner. A richer integration which supports e.g. alternative ways do recognize activities and preferences on those will be developed during years two and three. Finally, methods for deploying a configuration have been developed in **task 3.5 Deployment**; future work includes monitoring of configurations to detect e.g. malfunctioning devices.

A Tasks of WP3 accordining to DoW

Task 3.1 Long-term context recognition (OrU)

Context recognition must provide two capabilities in order to guarantee that relevant events/activities and trends are recognized in a timely fashion. First, data received from the sensor network needs to be interpreted and represented in symbolic form in order to allow inference mechanisms to assess whether specific conditions hold. This equates to bringing signal-level data (e.g., the readings of a proximity sensor, or raw physiological data) to a higher level of abstraction that indicates crisp states of the human being and of the environment. The second step is to actually perform inference in this information in order to obtain high-level descriptors of the state of the human user which can support human decision making (e.g., activities of daily living). The former problem of bridging the gap between signal-level data and symbols will be addressed using data-driven techniques like Hidden Markov Models, Neural Networks and other Machine Learning methods. The second step will be achieved through the use of constraint-based reasoning techniques. For instance, we want to be able to use information about the assisted person entering the kitchen (a primitive event which can be detected by sensors after some processing) as part of an inference that the user is having lunch (abstract activity). The benefit of using constraint-based techniques is twofold: first, it has been shown that these techniques scale well to long temporal horizons (i.e., the context recognition system can operate continuously, automatically inferring when "older" events should be discarded and dynamically adapting its search procedure to large amounts of data); a second benefit is that planning can be directly integrated in to the inference procedure, which entails that certain system's events such as reminders and warnings can be directly modeled into the specification of behavioural patterns, and their execution state can be monitored (e.g., recognizing if a person has not understood a warning and reacting to this contingency). A particular challenge that we will address is that particular correlations of physiological/environmental sensor readings should be defined in a redundant manner, i.e., there may be several partially overlapping criteria that indicate the same high-level state. For instance, the way in which blood sugar level should correlate to meal assumption is strongly dependent on the time of day, on the person, and on other physiological factors. More in general, context recognition will work under different conditions in different homes, or even in the same home at different points in time, as the set of available sensors and their deployment may vary from case to case. A second challenge that we will address is to combine data-driven methods for activity recognition as a "pre-processing" step for the symbolic reasoning layer. This will add significant capabilities to the system in terms of real-world deployability. To this end, we will employ probabilistic reasoning techniques at the symbolic level in order to heed against sensory uncertainty. Techniques we will investigate include Bayesian networks, which have been shown to integrate well with general constraint reasoning infrastructures. Also, in order to enhance the system's ability to recognize context under different conditions and evolving personal habits and behaviours, constraint-based context inference will be supported by selected machine learning techniques for activity recognition (e.g., [Wu et al., 2007, Patterson et al., 2005]). This will endow the system with sufficient self-learning capabilities for deployment in different contexts with different users. These techniques will also be used to develop simple self-adaptation to evolving personal habits of the users. Thus while constraint-based probabilistic contextual models will provide flexible and easily customizable models for context recognition and automated trend analysis, learning techniques employed on specific elements of these models will provide adaptation to small changes in behavior over time. Overall, this task will contribute important scientific contributions, including hybrid methods for context recognition and a more general suite of techniques which are not relegated to activity recognition, rather extend contextual inference to other aspects of human monitoring. The activity recognition system will issue tasks in terms of what primitive events to observe to the configuration planner (task 3.2). It will also be connected to the user interface in WP4. An important part of this task is the design and implementation of a contextual model in a representation. Parts of this contextual model will also be accessible by the various users of the Giraff+ system through the user interfaces developed in WP 4. A first prototype is delivered in M12 (D3.1), a second in M18 to be used at test sites (D3.2), a third with uncertainty management in M30 (D3.3), and a final in M36 (D3.4).

Task 3.2 Configuration planning for efficient activity recognition (OrU)

Given a certain set of activities and a set of available sensors and programs for extracting information from these sensors, the sensor network needs to be configured in such a way that it is capable of of inferring the activities from the sensor data. Such a configuration specifies what sensors to be used, and what programs will be used to extract information at different levels of abstraction from them. For this purpose, we will develop configuration planning techniques [Lundh et al., 2008], that can automatically find an optimal configuration for a given task (i.e. given set of activities of interest). This configuration must be based on "primitive events" from the perspective of activity recognition, and there might be several ways to detect these "primitive events". Hence, the problem of configuration planning includes inferring what primitive events are necessary and/or sufficient for recognizing the given activities. Configuration planning would permit to add or remove sensors from a site: the configuration planner would then find a new configuration based on the new set of sensors. Likewise, if the current configuration does not perform its tasks satisfactorily (e.g. because a sensor malfunctions), a configuration planner could find a better one. This task includes handling such cases. In addition, the configuration planner should ensure that the sensor system is used in an efficient manner and that sensors that are not needed are switched off. The configuration planner will take tasks as input from the context recognition module (task 3.1). It might also take tasks related to services requested by the user from the Client-side UI (WP 4). A first prototype is delivered in M12 (D3.1), a second in M18 at test sites (D3.2), a third with dynamic and multiple objectives and changing devices in M30 (D3.3), and a final in M36 (D3.4).

Task 3.3 Integration of context inference and configuration planning (OrU)

This task will focus on the integration of configuration planning into the constraint-based context inference framework developed in task 3.1. Specifically, work will proceed in two stages. First, the ability to specify hierarchical decompositions will be built into the constraint reasoning system. This will allow complementing the constraint language used for specifying correlations between sensor readings and human state with relevant information on which sensors should be activated and how they should be configured to maximize the information gathering process. Secondly, work will focus on developing hierarchical reasoning techniques on top of the temporal inference mechanism of the context inference and actuation infrastructure. This will result in a hybrid planning algorithm which can reason upon quantified temporal relationships while searching for appropriate configurations of sensors. The algorithm will employ multi-objective optimization criteria, whereby temporal constraint satisfability (which is the basis for context inference) will be enforced while (1) maximizing the likelihood of inferred context based on given measures of uncertainty (developed in task 2.1), and (2) optimizing the cost of sensing and acting in the environment (which is the main optimization criteria for configuration planning). A first prototype of the resulting system will be available in month 12. It will be integrated into the test sites by month 18 (see WP5), and further refined during the rest of the project to take into account the requirements stemming from evaluations with users. A third prototype is delivered in M30 (D3.3), and a final in M36. (D3.4).

Task 3.5 Deployment (OrU, CNR-ISTC, MDH)

In connection with the integration tasks of WP5, this task aims to design and implement routines for taking a description of a configuration generated by the configuration planner and start and connect the actual sensors, devices, interfaces and other programs specified in that description. In addition, failures in components and communication as well as successful task completions should be detected and responded to by reconfiguration.

References

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.

James Frederick Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, November 1983. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/182.358434. URL `http://doi.acm.org/10.1145/182.358434`.

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.

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.

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`.

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. URL `http://dl.acm.org/citation.cfm?id=1625275.1625326`.

G. Duftschmid, S. Miksch, and Gall. Verification of temporal scheduling constraints in clinical practice guidelines. *Art. Intelligence in Medicine*, 25(2): 93–121, 2002.

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`.

Ronald Lewis Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

Günther Greiner and Kai Hormann. Efficient clipping of arbitrary polygons. *ACM Transactions on Graphics*, 17(2):71–83, April 1998.

M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26(1):191–246, 2006.

Stefan Hertel and Kurt Mehlhorn. Fast triangulation of the plane with respect to simple polygons. *Information and Control*, 64(1-3):52–76, 1985.

J. Mark Keil. Decomposing a polygon into simpler components. *SIAM J. Comput.*, 14(4):799–817, 1985.

W. Liu and W. Havens. Histogram arc consistency as a value ordering heuristic. *Advances in Artificial Intelligence*, pages 510–516, 2004.

R. Lundh, L. Karlsson, and A. Saffiotti. Automatic configuration of multi-robot systems: Planning for multiple steps. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-08)*, 2008.

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`.

M. Mansouri. Constraint-Based Activity Recognition with Uncertainty. Master's thesis, Örebro University, School of Science and Technology, 2011.

D. Patterson, D. Fox, H. Kautz, and M. Philipose. Fine-grained activity recognitionby aggregating abstract object usage. In *Proc. of the 9th IEEE International Symposium on Wearable Computers*, 2005.

F. Pecora, M. Cirillo, F. Dell'Osa, J. Ullberg, and A. Saffiotti. A Constraint-Based Approach for Proactive, Context-Aware Human Support. *Journal of Ambient Intelligence and Smart Environments*, 2012. (accepted).

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.

Jean-Francois Rit. Propagating temporal constraints for scheduling. In *Proceedings of the 5th National Conference on Artificial Intelligence*, 1986.

Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120:248–253, 1998.

E. Tsang. *Foundations of constraint satisfaction.* Computation in cognitive science. Academic Press, 1993. ISBN 9780127016108. URL `http://books.google.se/books?id=TnxQAAAAMAAJ`.

Jonas Ullberg, Amy Loutfi, and Federico Pecora. Towards Continuous Activity Monitoring with Temporal Constraints. In *Proceedings of the 4th Workshop on Planning and Plan Execution for Real-World Systems at ICAPS09*, 2009.

Jonas Ullberg, , and Federico Pecora. Propagating Constraints on Sets of Intervals. In *ICAPS Workshop on Planning and Scheduling with Timelines (PSTL)*, 2012.

Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4), 1994.

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.