A Constraint-Based Approach for Plan Management in Intelligent Environments*

Federico Pecora and **Marcello Cirillo**
Center for Applied Autonomous Sensor Systems
Örebro University, SE-70182 Sweden
<name>.<surname>@oru.se

Abstract

In this paper we address the problem of realizing a service-providing reasoning infrastructure for proactive human assistance in intelligent environments. We propose SAM, an architecture which leverages temporal knowledge represented as relations in Allen's interval algebra and constraint-based temporal planning techniques. SAM seamlessly combines two key capabilities for contextualized service provision, namely human activity recognition and planning for controlling pervasive actuation devices.

Introduction

The problem we tackle in this paper is that of realizing a service-providing reasoning infrastructure for proactive human assistance in intelligent environments. Two key capabilities that are often desirable in a service-providing intelligent environment are (1) the ability to recognize activities performed by the human user, and (2) the ability to plan and execute the behavior of pervasive service-providing devices according to the indications of activity recognition.

Activity recognition has received much attention in the literature and the term has been employed to indicate a variety of capabilities. In this paper we take activity recognition to mean the ability of the intelligent system to deduce temporally contextualized knowledge regarding the state of the user on the basis of a set of heterogeneous sensor readings. Equipped with such a capability, an intelligent environment could be capable of proactively planning for and executing services that provide contextualized assistance. This requires a way to model the temporal and causal dependencies that exist between these tasks and the state of the human user. For instance, if a smart home could recognize that the human user is cooking, it could instruct a cleaning robot to avoid navigating to the the dining room until the subsequent dining activity is over.

This paper presents SAM, an Activity Management architecture[1] for service providing intelligent environments which achieves the two key capabilities mentioned above. SAM is built on top of the Multi-component Planning and Scheduling framework (OMPS) (Fratini, Pecora, and Cesta 2008). Specifically, in conjunction with an intelligent environment equipped with pervasive sensors and actuators, SAM provides the means to monitor the daily activities of a human being and to proactively assist the human through the environment's actuators. The architecture realizes an on-line abductive reasoning process on patterns of sensor observations provided by the intelligent environment, and is capable of synthesizing action plans for the environment's actuators in reaction to recognized human activities. As a direct result of the underlying framework, SAM retains three important properties: (1) the component-based domain description language provides a common formalism for expressing the activity recognition and proactive controller functionalities of the domain; (2) the constraint-based nature of the architecture allows to perform concurrent activity recognition, planning and execution; (3) the component-based nature of the framework allows to implement modular interfaces to the intelligent environment, thus supporting the incremental integration of new sensory/actuation elements.

Related Work

Current approaches to the problem of recognizing human activities can be roughly categorized as *data-driven* or *knowledge-driven*. In data-driven approaches, models of human behavior are learned from large volumes of data over time. Notable examples of this approach employ Hidden Markov Models (HMMs) for learning sequences of sensor observations with given transition probabilities, e.g., (Wu et al. 2007). Knowledge-driven approaches follow a complementary approach in which patterns of observations are modeled from first principles rather than learned. Such approaches typically employ an abductive processes, whereby sensor data is explained by hypothesizing the occurrence of specific human activities. Examples include reasoning approaches in which rich temporal representations are employed to model the conditions under which patterns of human activities occur (Jakkula, Cook, and Crandall 2007).

Data- and Knowledge-driven approaches have complementary strengths: the former provide an effective way to recognize elementary activities from large amounts of continuous data; conversely, knowledge-driven approaches are

*This paper appears in the *Proceedings of the Scheduling and Planning Applications Workshop (SPARK)*, held in conjunction with the 19th International Conference on Planning and Scheduling, 2009.

[1] SAM stands for "SAM the Activity Manager".

useful when the criteria for recognizing human activities are given by crisp rules that are clearly identifiable. In SAM, we follow the latter approach.

Also relevant to our work are various uses of schedule execution monitoring techniques for domestic activity monitoring presented in the literature, e.g., (Cesta et al. 2007; Pollack et al. 2003). An important difference with the above works lies in the fact that they employ pre-compiled (albeit highly flexible) schedules as models for human behavior. In the present work, we employ a planning process to actually instantiate such candidate schedules on-line.

SAM leverages the capability of OMPS to plan for state variables, a feature typical of several continuous planning approaches (Knight et al. 2001). In addition, SAM leverages the ability of OMPS to employ custom variable types. This has allowed us to build the sensing and actuation capabilities directly into new variable types which extend the state variable. In SAM, variables are not only used to represent elements of the domain, but also to implement active processes which operate concurrently with the continuous planning process, providing it with real world data obtained form the intelligent environment.

Lastly, SAM is related to the situation recognition approach described in (Dousson, Gaborit, and Ghallab 1993), which also employs temporal reasoning techniques to perform on-line recognition of temporal patterns of sensory events. Like SAM, the requirements for recognition are modeled as temporal relations in Allen's interval algebra, and both recognition and actuation are modeled within the same formalism. However, in SAM these two types of reasoning are integrated at the reasoning level in addition to being described by the same formalism. Also, while the former approach is limited to "triggering" events as a result of recognized situations, SAM allows to trigger the generation of a contingent plan whose elements are flexibly constrained to sensory events or recognized activities as they evolve in time.

Domain Representation

SAM is implemented within the OMPS temporal reasoning framework (Fratini, Pecora, and Cesta 2008). OMPS is a constraint-based planning and scheduling software API for developing temporal planning and scheduling applications, and has been used to develop a variety of decision support tools, ranging from highly-specialized space mission planning software to classical planning frameworks.

SAM leverages the domain description language provided by OMPS to model the dependencies that exist between sensor readings, the state of the human user, and tasks to be performed in the environment. In this section we describe how domains expressed in this formalism can be used to represent both requirements on sensor readings and on actuation devices. The following section will describe the actual implementation of SAM, i.e., how such domain descriptions are employed to infer the state of the user and to contextually synthesize action plans for actuators in the intelligent environment.

OMPS's domain description language is grounded on the notion of *component*. A component is an element of a domain theory which represents a logical or physical entity. Components model parts of the real world that are relevant for a specific decisional process, such as complex physical systems or their parts. Components can be used to represent, for example, a robot which can navigate the environment and grasp objects, or an autonomous refrigerator which can autonomously open and close its door.

An automated reasoning functionality developed in OMPS consists in a procedure for taking *decisions* on components. Decisions describe an assertion on the possible evolutions in time of a component. For instance, a decision on the fridge component described above could be to open its door no earlier than time instant 30 and no later than time instant 40. More precisely, a decision is an assertion on the value of a component in a given flexible time interval, i.e., a pair $\langle v, [I_s, I_e] \rangle$, where the nature of the value v depends on the specific component and I_s, I_e represent, respectively, an interval of admissibility of the start and end times of the decision. In the fridge example, assuming the door takes five seconds to open, the flexible interval is $[I_s = [30, 40], I_e = [34, 44]]$.

OMPS provides a number of built-in component types, among which consumable and re-usable multi-capacity resources, and state variables. The built-in state variable type of component instead models elements whose state in time is represented by a symbol. OMPS supports disjunctive values for state variables, e.g., a decision on a state variable that models a mobile robot could be $\langle \mathbf{navigate} \vee \mathbf{grasp}, [I_s, I_e] \rangle$, representing that the robot should be in the process of either navigating or grasping an object during the flexible interval $[I_s, I_e]$. For the purposes of this work, we focus on state variable type components and two custom components that have been developed in SAM to accommodate the needs of the physically instantiated nature of our application domain.

The core intuition behind OMPS is the fact that decisions on certain components may entail the need to assert decisions on other components. For instance, the decision to dock the robot to the fridge may *require* that the fridge door has already been opened. Such dependencies among component decisions are captured in a domain theory through what are called *synchronizations*. A synchronization is a set of requirements expressed in the form of temporal constraints. Such constraints in OMPS are bounded variants of the relations in the restricted Allen's Interval Algebra (Allen 1984; Vilain, Kautz, and van Beek 1989). Specifically, temporal constraints in OMPS enrich Allen's relations with bounds through which it is possible to fine-tune the relative temporal placement of constrained decisions. For instance, the constraint **A** DURING $[3, 5][0, \infty)$ **B** states that **A** should be temporally contained in **B**, that the start time of **A** must occur between 3 and 5 units of time after the beginning of **B**, and that the end time of **A** should occur some time before the end of **B**.

Figure 1(a) shows an example of how temporal constraints can be used to model requirements among actuators in an intelligent environment. The three synchronizations involve two components: a robotic table and an intelligent fridge (represented, respectively, by state variables

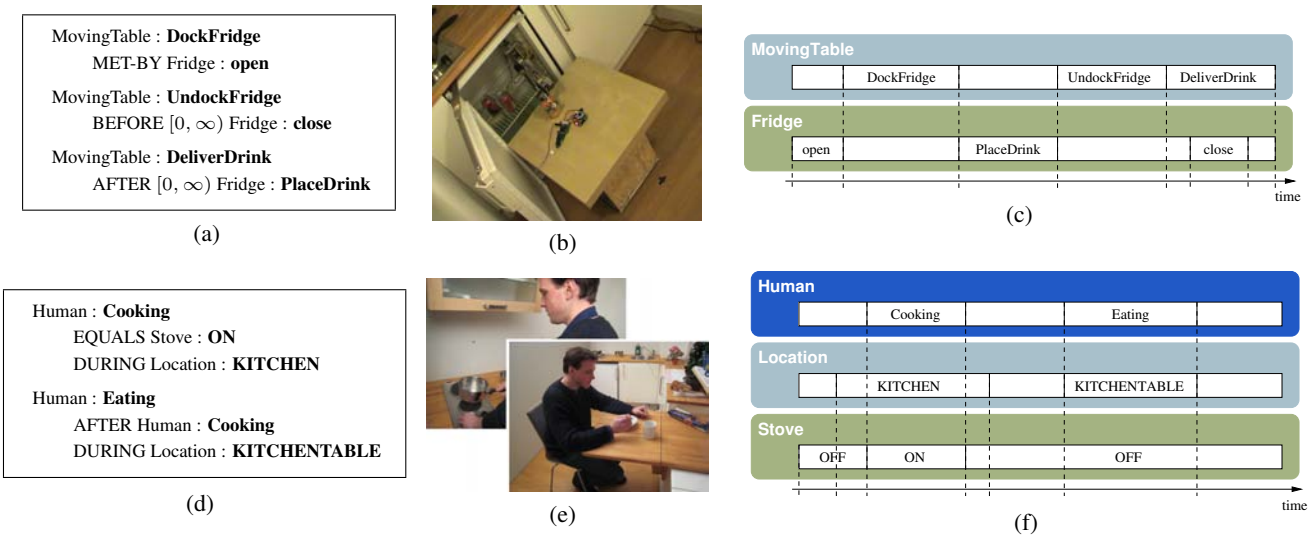

Figure 1: *Top row:* three synchronizations in a possible domestic robot planning domain (a), the corresponding real components available in our intelligent environment (b), and a possible timeline for the two components (c). *Bottom row:* two synchronizations in a possible domestic activity recognition domain (d), the corresponding situations as enacted by a test subject in a test environment (e), and a possible timeline for the three components (f).

MovingTable and Fridge). The MovingTable can dock and undock the Fridge, and navigate to the human user to deliver a drink. The Fridge component can open and close its door, as well as grasp a drink inside it and place it on a docked table. The above three synchronizations model three simple requirements of this domain, namely: (1) since the Fridge's door cannot open if it is obstructed by the MovingTable (see figure 1(b)), and we would like the door to be kept open only when necessary, docking the fridge must occur directly after the fridge door is opened (MET-BY constraint); (2) for the same reasons, the fridge door should close only after the MovingTable has completed the undocking procedure (BEFORE constraint); and (3) delivering a drink to the human is possible only after the drink has been placed on the table (AFTER constraint).

While temporal constraints express requirements on the temporal intervals of decisions, value constraints express requirements on the value of decisions. OMPS provides the VALUE-EQUALS constraint to model that two decisions should have equal value. For instance, asserting d_1 VALUE-EQUALS d_2 where the two decisions' values are, respectively, $\mathbf{v_1} = \mathbf{A} \vee \mathbf{B}$ and $\mathbf{v_2} = \mathbf{B} \vee \mathbf{C}$, will constrain the value of both decisions to be $\mathbf{B}$ (the intersection of possible values). As for temporal constraints, OMPS provides built-in propagation for value constraints.

Decisions and temporal constraints asserted on components are maintained in a *decision network* (DN), that is at all times kept consistent through *temporal propagation*. This ensures that the temporal intervals underlying the decisions are kept consistent with respect to the temporal constraints, while decisions are anchored flexibly in time. In other words, adding a temporal constraint to the DN will either result in the calculation of updated *bounds* for the intervals I_s, I_e for all decisions, or in a propagation failure, indicating that the added constraint or decision is not admissible. Temporal constraint propagation is a polynomial time operation, as it is based on a Simple Temporal Network (Dechter, Meiri, and Pearl 1991).

For each component in the domain, OMPS provides built-in methods to extract the *timeline* of the component. A timeline represents the behavior of a component in time as it is determined by the decisions and constraints imposed on this component in the DN. Figure 1(c) shows a possible timeline for the two components Fridge and MovingTable of the previous example. Notice that, in general, it is possible to extract many timelines for a component, as constraints bound decision start and end times flexibly. In the remainder of this paper we will always employ the earliest start time timeline, i.e., the timeline obtained by choosing the lower bound for all decisions' temporal intervals I_s, I_e.

In the previous example temporal constraints are used to model the requirements that exist between two "actuator components" (modeled as state variables) in carrying out the task of retrieving a drink from the fridge. In addition to actuators, however, state variables can be used to represent sensors in an intelligent environment, their values thus representing *sensor readings* rather than commands to be executed. Consequently, while temporal constraints among the values of actuator components represent temporal dependencies among commands to be executed that should be upheld in proactive service enactment, temporal constraints among "sensor components" represent temporal dependencies among sensor readings that are the result of specific human activities. For instance, the synchronizations in figure 1(d) describe possible conditions under which the human activities of **Cooking** and **Eating** can be inferred (where omitted, temporal bounds are assumed to be $[0, \infty)$). The synchronizations involve three components, namely a state variable representing the human inhabitant of the intelligent environment, a state variable representing a stove state sen-

sor, and another state variable representing the location of the human as it is determined by a person localization sensor in the environment. The synchronizations model how the relative occurrence of specific values of these components in time can be used as evidence of the human cooking or eating: the former is deduced as a result of the user being located in the **KITCHEN** (DURING constraint) and is temporally equal to the sensed activity of the Stove sensor; similarly, the requirement for asserting the **Eating** activity consists in the human being having already performed the **Cooking** activity (AFTER constraint) and his being seated at the **KITCHENTABLE**.

A unique feature of SAM is that the same formalism can be employed to express requirements both for enactment and for activity recognition. This is enabled by two specializations of the state variable component type, namely *sensor components* and *actuator components*. As we will see, a single inference algorithm based on temporal constraint reasoning provides a means to concurrently deduce context from sensor components and to plan for actuator components.

Recognizing Activities and Executing Proactive Services in SAM

SAM employs three types of components: *state variables*, *sensors* and *actuators*. State variables are employed to model one or more aspects of the user's activities of daily living. For instance, in the examples that follow we will use a state variable whose values are $\{\mathbf{Cooking}, \mathbf{Eating}, \mathbf{InBed}, \mathbf{WatchingTV}, \mathbf{Out}\}$ to model the human user's domestic activities. Sensors and actuators are specialized variants of the built-in state variable type which implement an interface between the real-world sensing and actuation modules and the DN. Sensor components interpret data obtained from the physical sensors deployed in the intelligent environment and represent this information as decisions and constraints in the DN. Actuators are components that trigger the execution on a real actuator of a planned decision. Actuators also have a sensing capability which allows to update the DN with relations that model the temporal bounds of execution of the executed operations.

In SAM, the DN acts as a "blackboard" where decisions and constraints re-construct the reality observed by sensor components as well as the current hypothesis on what the human being is doing. This hypothesis is deduced by a continuous re-planning process which attempts to infer new possible states of the human being and any necessary actuator plans.

SAM is implemented as a multitude of concurrent processes (described in detail in the following sections), each operating continuously on the DN:

Sensing processes: each sensor is a process that adds decisions and constraints to represent the real-world observations provided by the intelligent environment.

Inference process: the current DN is manipulated by the continuous inference process, which adds decisions and constraints that model the current activity performed by the user and any proactive support operations to be executed by the actuators.

Actuator processes: actuators ensure that decisions in the DN that represent operations to be executed are dispatched as commands to the real actuators and that termination of actuation operations are reflected in the DN as they are observed in reality.

These processes add decisions and constraints to the DN in real-time, and access to the DN is scheduled by an overall process scheduler. Each process modifies the DN, thus triggering constraint propagation.

Continuous Inference Process

SAM's continuous inference process relies on the fact that the DN represents at all times the current situation in the real world, possesses two key capabilities: (1) to assess whether the DN contains evidence of sensed values in a given time interval; and (2) to assess whether the DN contains the requirements described in a particular synchronization. Both capabilities can be viewed as ways to *support* candidate decisions. Supporting a decision means performing one of the two following steps:

Unification. A decision is supported by unification if it is possible to impose a temporal EQUALS constraint and a VALUE-EQUALS constraint between it and another decision which is already supported. If the result of imposing these two constraints is successful, then this is an indication that indeed there is an interval of time in which the value of the decision to support has been sensed in the real environment. SAM can therefore "query" the DN to assess whether a value $\mathbf{v}$ has been sensed in a certain interval of time $[I_s, I_e]$ by attempting to support through unification a decision $\langle \mathbf{v}, [I_s, I_e] \rangle$.

Expansion. Expanding a synchronization entails that new (unsupported) decisions and constraints are added to the DN as prescribed by the requirements of the synchronization. Support for these new decisions is sought by recursively expanding other synchronizations or unifying the new decisions with others already present in the DN. Overall, expansion is how SAM assesses whether the current situation of sensor readings in the DN can support a particular hypothesis: it adds an unsupported decision representing the current hypothesis (e.g., that the human being is cooking), and tries to support it through the domain theory and existing sensed values in the DN.

The continuous re-planning process implemented in SAM is shown in procedure `Replan`. The procedure leverages unification and expansion to continuously attempt to support decisions which represent hypotheses on the state of a number of *monitored* components. These components are all those components for which we wish SAM to deduce their current state. In our specific application domain, all these components are state variables which model some aspect of the human user's state. For each monitored component, the procedure adds to the DN a decision whose value is a disjunction of all its possible values (lines 2–3). For instance, if the component in question is the state variable Human described previously, then the new decision to be added will be $d_{hyp}^{\text{Human}} = \langle (\mathbf{Cooking} \vee \mathbf{Eating} \vee \mathbf{InBed} \vee$

$\mathbf{WatchingTV} \vee \mathbf{Out}), [[0,\infty),[0,\infty)]\rangle$. This decision is marked as un-supported (line 4), i.e., it constitutes a hypothesis on the current activity in which the human user is engage in. The procedure then constrains this decision to occur after any other decisions on the same component (lines 5–6). This is done in order to avoid that the new decision is trivially supported by unification with a decision supported in a previous call to the `Replan` procedure. Finally, the procedure triggers a decision supporting algorithm which attempts to support the newly added decisions by recursively expanding synchronizations and unifying the resulting requirements (line 7). In the process of supporting new decisions, their values will be constrained (by VALUE-EQUALS constraints) to take on a specific value. For instance, if the domain theory contains a synchronization stating that the requirements for **Eating** on component Human are a certain set of values on some sensor components, then the unsupported decision is marked as supported, the unary constraint d_{hyp}^{Human} VALUE-EQUALS **Eating** is imposed, and new (un-supported) decisions on the sensor components are added to the DN.

Procedure `Replan(`*DN*`)`

1 **foreach** $c \in$ *MonitoredComponents* **do**
2 $\quad \mathbf{v} \leftarrow \bigvee_{v_i \in \text{possibleValues}(c)} v_i$
3 $\quad$ DN $\leftarrow$ DN $\cup\ d_{hyp}^{c} = \langle \mathbf{v}, [I_s, I_e] \rangle$
4 $\quad$ mark d_{hyp}^{c} as not supported
5 $\quad$ **foreach** d *on component* c **do**
6 $\quad\quad$ DN $\leftarrow$ DN $\cup\ d_{hyp}^{c}$ AFTER $[0,\infty)\ d$
7 `SupportDecisions(`*DN*`)`

If the decision supporting algorithm terminates successfully, the DN contains the new decisions that have been added by the re-planning procedure, plus all those decisions and constraints that implement support for these decisions. The value of each newly supported decision on monitored components has been constrained to be that required by the synchronization that was used by the `SupportDecisions` procedure. Since these decisions are linked by temporal constraints to decisions on sensor components, their placement in time will follow the evolution of the DN's decisions on sensor components as time progresses.

If `SupportDecisions` fails, the resulting DN is identical to before the re-planning procedure was started, therefore reflecting the fact that no new information was deduced.

Note that the continuous `SupportDecisions` procedure is greedy, in that the first successfully applicable synchronization is selected in support of current sensor readings.

Sensing Processes

In order to realize the interface between OMPS and real-world sensors in the intelligent environment, a new component, the *sensor*, was developed in SAM. A sensor is modeled in the domain for each physical sensor in the intelligent environment. Each sensor component is provided with an interface to the physical sensor, as well as the capability to periodically update the DN with decisions and constraints that model the state of the physical sensor. The process for updating the DN is described in procedure `UpdateSensorValues`. Specifically, each sensor component's sensing procedure obtains from the DN the decision that represents the value of the sensor at the previous iteration (line 1). This decision, if it exists, is the decision whose end time has an infinite upper bound (u_e). No such decision exists if at the previous iteration the sensor readings were undetermined (d is null, i.e., there is no information on the current sensor value in the DN). The procedure then obtains the current sensor reading from its interface to the physical sensor (line 2). Notice that this could also be undetermined (null in the procedure), as a sensor may not provide a reading at all. At this point, three situations may occur.

Procedure `UpdateSensorValues(`DN,t_{now}`)`

1 $d \leftarrow \langle \mathbf{v}, [[l_s, u_s], [l_e, u_e]] \rangle \in$ DN $\ s.t.\ u_e = \infty$
2 $\mathbf{v_s} \leftarrow$ `ReadSensor()`
3 **if** $d = \text{null} \ \wedge\ \mathbf{v_s} \neq \text{null}$ **then**
4 $\quad$ DN $\leftarrow$ DN $\cup\ d' = \langle \mathbf{v_s}, [[0,\infty),[0,\infty)] \rangle$
5 $\quad$ DN $\leftarrow$ DN $\cup\ d'$ RELEASE $[t_{\text{now}}, t_{\text{now}}]$
6 $\quad$ DN $\leftarrow$ DN $\cup\ d'$ DEADLINE $[t_{\text{now}}+1, \infty]$
7 **else if** $d \neq \text{null} \ \wedge\ \mathbf{v_s} = \text{null}$ **then**
8 $\quad$ DN $\leftarrow$ DN $\cup\ d$ DEADLINE $[t_{\text{now}}, t_{\text{now}}]$
9 **else if** $d \neq \text{null} \ \wedge\ \mathbf{v_s} \neq \text{null}$ **then**
10 $\quad$ **if** $\mathbf{v_s} = \mathbf{v}$ **then**
11 $\quad\quad$ DN $\leftarrow$ DN $\cup\ d$ DEADLINE $[t_{\text{now}}+1, \infty]$
12 $\quad$ **else**
13 $\quad\quad$ DN $\leftarrow$ DN $\cup\ d$ DEADLINE $[t_{\text{now}}, t_{\text{now}}]$
14 $\quad\quad$ DN $\leftarrow$ DN $\cup\ d' = \langle \mathbf{v_s}, [[0,\infty),[0,\infty)] \rangle$
15 $\quad\quad$ DN $\leftarrow$ DN $\cup\ d'$ RELEASE $[t_{\text{now}}, t_{\text{now}}]$
16 $\quad\quad$ DN $\leftarrow$ DN $\cup\ d'$ DEADLINE $[t_{\text{now}}+1, \infty]$

New sensor reading. If the DN does not contain an unbounded decision and the physical sensor returns a value, then a decision is added to the DN representing this (new) sensor reading. The start time of this decision is anchored to the current time t_{now} by means of a RELEASE constraint and made to have an unbounded end time (lines 3–6). If the DN contains an unbounded decision that differs from the sensor reading, then the procedure models this fact in the DN as above, and in addition "stops" the previous decision by imposing a DEADLINE constraint, i.e., anchoring the decision's end time to t_{now} (lines 9, 12–16).

Continued sensor reading. If the DN contains an unbounded decision and the physical sensor returns the same value as that of this decision, then the procedure ensures that the increased duration of this decision is reflected in the DN. It does so by updating the lower bound of the decision's end time to beyond the current time by means of a new DEADLINE constraint (lines 9–11). Notice that this ensures that at the next iteration the DN will contain an unbounded decision.

Interrupted sensor reading. If the DN contains an unbounded decision and the physical sensor returns no read-

ing ($\mathbf{v_s}$ is null), then the procedure simply interrupts the unbounded decision by bounding its end time to the current time with a DEADLINE constraint (lines 7–8).

Actuation Processes

The inference procedure implemented in SAM continuously assesses the applicability of given synchronizations in the current DN by asserting and attempting to support new decisions on monitored components, such as the Human state variable presented earlier. This same mechanism allows to obtain contextualized plan synthesis capabilities through the addition of synchronizations that model how actions carried out by actuators should be temporally related to recognized activities. For instance, in addition to requiring that **Cooking** should be supported by requirements such as "being in the kitchen" and "using the stove", a requirement involving an actuator component can be added, such as "turn on the ventilation over the stove". More in general, for each actuation-capable device in the intelligent environment, an actuator component is modeled in the domain. This component's values represent the possible commands that can be performed by the device. In the domain, these values are added as requirements to the synchronizations of monitored components. As sensor components interface the real world to represent sensor readings in the DN, actuator components interface the real world to trigger commands to real actuators when decisions involving them appear in the DN.

However, it should be noticed that robotic devices are only partially controllable, in that we do not have strict guarantees on when and for how long given commands will be executed. For this reason, actuator components also possess a sensory capability that is employed to feed information on the status of command execution back into the DN. As sensor components, actuator components write this information directly into the DN, thus allowing the re-planning process to take into account the current state of execution of the actions.

Procedure `UpdateExecutionState(`DN,t_{now}`)`

1 $D \leftarrow \{\langle \mathbf{v}, [[l_s, u_s], [l_e, u_e]]\rangle \in \text{DN} : l_s \leq t_{\text{now}}, u_e = \infty\}$
2 **foreach** $d \in D$ **do**
3 **if** `IsExecuting(`$\mathbf{v}$`)` **then**
4 DN $\leftarrow$ DN $\cup$ d DEADLINE $[t_{\text{now}} + 1, \infty]$
5 **else if** $l_s = l_e$ **then**
6 `StartExecuting(`$\mathbf{v}$`)`
7 DN $\leftarrow$ DN $\cup$ d RELEASE $[t_{\text{now}}, t_{\text{now}}]$
8 **else** DN $\leftarrow$ DN $\cup$ d DEADLINE $[t_{\text{now}}, t_{\text{now}}]$

Actuators execute concurrently with the re-planning and sensing operations described above. The operations performed by actuators are shown in procedure `UpdateExecutionState`. Each actuator component first identifies all decisions that have an unbounded end time and whose earliest start time falls before or at the current time (line 1). The fact that these decisions are unbounded indicates that they have been planned for execution and their execution has not yet terminated. The fact that their start time lies before or at the current time indicates that they are scheduled to start or have already begun. For each of these decisions, the physical actuator is queried to ascertain whether the corresponding command is being executed. If so, then the decision is constrained to end at least one time unit beyond the current time (lines 3–4). If the command is not currently in execution, the procedure checks whether the command still needs to be issued to the physical actuator. This is the case if the earliest start and end times of the decision coincide (because the decision's end time was never updated at previous iterations). The procedure dispatches the command to the actuator and anchors the start time of the decision to the current time (lines 5–7). Conversely, if the decision's start and end times do not coincide, then the decision is assumed to be ended, and the procedure imposes the current time as its earliest and latest end time (line 8).

Case Studies in the PEIS-Home

We illustrate the use of sensor components in SAM with four runs performed in the PEIS-Home, a prototypical intelligent environment deployed at the at Örebro University (see `aass.oru.se/~peis`). The environment provides ubiquitous sensing and actuation devices, including the robotic table and intelligent fridge described in earlier examples.

In the first run our aim is to assess the sleep quality of a person by tracking how many times and for how long the user turns on his night light when he lies in bed. For this purpose, we employ three physical sensors: a pressure sensor, placed beneath the bed, a luminosity sensor placed close to the night light, and a person tracker based on stereo vision. We then define a domain with three sensor components and the two synchronizations shown in figure 2. Note

1) Human : **InBed**	2) HumanAbstract : **Awake**
DURING Location : **NOPOS**	DURING Human : **InBed**
EQUALS Bed : **ON**	EQUALS NightLight : **ON**

Figure 2: Synchronizations defined in our domain for the Human and HumanAbstract components to assess quality of sleep.

that the human user is modeled by means of two distinct components, Human and HumanAbstract. This allows us to reason at different levels of abstraction on the user: while the decisions taken on component Human are always a direct consequence of sensor readings, synchronizations on values of HumanAbstract describe knowledge that can be inferred from sensor data as well as previously recognized Human and HumanAbstract activities. The first synchronization models two requirements for recognizing that the user has gone to bed: first, the user should not be observable by the tracking system, since the bedroom is a private area of the apartment and, therefore, outside the field of view of the cameras; second, the pressure sensor beneath the bed should be activated. The resulting **InBed** decision has a duration EQUAL to the one of the positive reading of the bed sensor. The second synchronization grasps the situation in which, although lying in bed, the user is not sleeping. The decision **Awake** on the component HumanAbstract depends therefore on the decision **InBed** of the Human and on the sensor readings of NightLight.

This simple domain was employed to test SAM in our intelligent home environment with a human subject. The overall duration of the experiment was 500 seconds, with the concurrent inference and sensing processes operating at a rate of about 1 Hz. Figure 5 (a) is a snapshot of the five components' timelines at the end of the run (from top to bottom, the three sensors and the two monitored components).

1) HumanAbstract : **Lunch**
STARTED-BY Human : **Cooking**
FINISHED-BY Human : **Eating**
DURING Time : **afternoon**

2) HumanAbstract : **Nap**
AFTER HumanAbstract : **Lunch**
EQUALS Human : **WatchingTV**

3) Human : **Cooking**
DURING Location : **KITCHEN**
EQUALS Stove : **ON**

4) Human : **WatchingTV**
EQUALS Location : **COUCH**

5) Human : **Eating**
DURING Location : **KITCHENTABLE**
EQUALS KTRfid : **DISH**

Figure 3: Synchronizations modeling afternoon activities of the human user.

The outcome of a more complex example is shown in figure 5 (b). In this case the scenario contains four instantiated sensors. Our goal is to determine the afternoon activities of the user living in the apartment, detecting activities like **Cooking**, **Eating** and the more abstract **Lunch**. To realize this example, we define five new synchronizations (figure 3), three for the Human component and two for the HumanAbstract component. Synchronization (3) identifies the human activity **Cooking**: the user should be in the kitchen and its duration is EQUAL to the activation of the Stove. Synchronization (5) models the **Eating** activity, using both the Location sensor and an RFID reader placed beneath the kitchen table (component KTRfid). A number of objects have been tagged to be recognized by the reader, among which dishes whose presence on the table is required to assert the decision **Eating**. The last synchronization for the Human component (4) correlates the presence of the user on the couch with the activity of **WatchingTV**.

Synchronizations (1) and (2) work at a higher level of abstraction. The decisions asserted on HumanAbstract are inferred from sensor readings (Time), from the Human component and from the HumanAbstract component itself. This way we can identify complex activities such as **Lunch**, which encompasses both **Cooking** and the subsequent **Eating**, and we can capture the fact that after lunch the user, sitting in front of the TV, will most probably fall asleep.

Also this example was executed in the PEIS-Home. It is worth mentioning that the decision corresponding to the **Lunch** activity on the HumanAbstract component was identified only when both **Cooking** and **Eating** were asserted on the Human component. Also it can be noted that **Nap** is identified as the current HumanAbstract activity only after the lunch is over and that on the first occurrence of **WatchingTV**, **Nap** was not asserted because it lacked support form the **Lunch** activity.

As an example of how the domain can include actuation as synchronization requirements on monitored components, let us consider the following run of SAM in a setup which includes the robotic table and autonomous fridge devices described earlier.

1) Human : **WatchingTV**
EQUALS Location : **COUCH**
START MovingTable : **DeliverDrink**

2) MovingTable : **DockFridge**
MET-BY Fridge : **OpenDoor**

3) MovingTable : **DeliverDrink**
AFTER Fridge : **PlaceDrink**

4) MovingTable : **UndockFridge**
BEFORE Fridge : **CloseDoor**

5) Fridge : **PlaceDrink**
MET-BY MovingTable : **DockFridge**
MEETS MovingTable : **UndockFridge**

6) OpenDoor : **OpenDoor**
MET-BY Fridge : **GraspDrink**

Figure 4: Synchronizations defining temporal relations between human activities and proactive services.

As shown in figure 4, we use abductive reasoning to infer when the user is watching TV. In this case, however, we modify the synchronization (4) presented figure 3 to include the actuators in the loop. The new synchronization (figure 4, (1)), not only recognizes the **WatchingTV** activity, but also asserts the decision **DeliverDrink** on the MovingTable component. This decision can be supported only if it comes AFTER another decision, namely **PlaceDrink** on component Fridge (synchronization (3)). When SAM's replanning procedure attempts to support **WatchingTV**, synchronization (5) is called into play, stating that **PlaceDrink** should occur right after (MET-BY) the MovingTable has docked the Fridge and right before the undocking maneuver (MEETS). The remaining three synchronizations — (2), (4) and (6) — are attempted to complete the chain of support, that is, the Fridge should first grasp the drink with its robotic arm, then open the door before the MovingTable is allowed to dock to it, and finally it should close the door right after the MovingTable has left the docking position.

This chain of synchronizations leads to the presence in the DN of a plan to retrieve a drink from the fridge and deliver it to the human who is watching TV. Notice that when the planned decisions on the actuator components are first added to the DN, their duration is minimal. Through the actuators' `UpdateExecutionState` procedure, these durations are updated at every re-planning period until the devices that are executing the tasks signal that execution has completed. Also, thanks to the continuous propagation of the constraints underlying the plan, decisions are appropriately delayed until their earliest start time coincides with the current time. A complete run of this scenario was performed in our intelligent environment and a snapshot of the final timelines is shown in figure 5 (c).

Conclusions and Future Work

In this paper we have presented SAM, an architecture for concurrent activity recognition, planning and execution. The architecture builds on the OMPS temporal reasoning framework, and leverages its component-based approach to realize a decisional framework that operates in a closed loop with physical sensing and actuation components in an intelligent environment. We have demonstrated the feasibility of the approach with a number of experimental runs in a real environment with a human test subject.

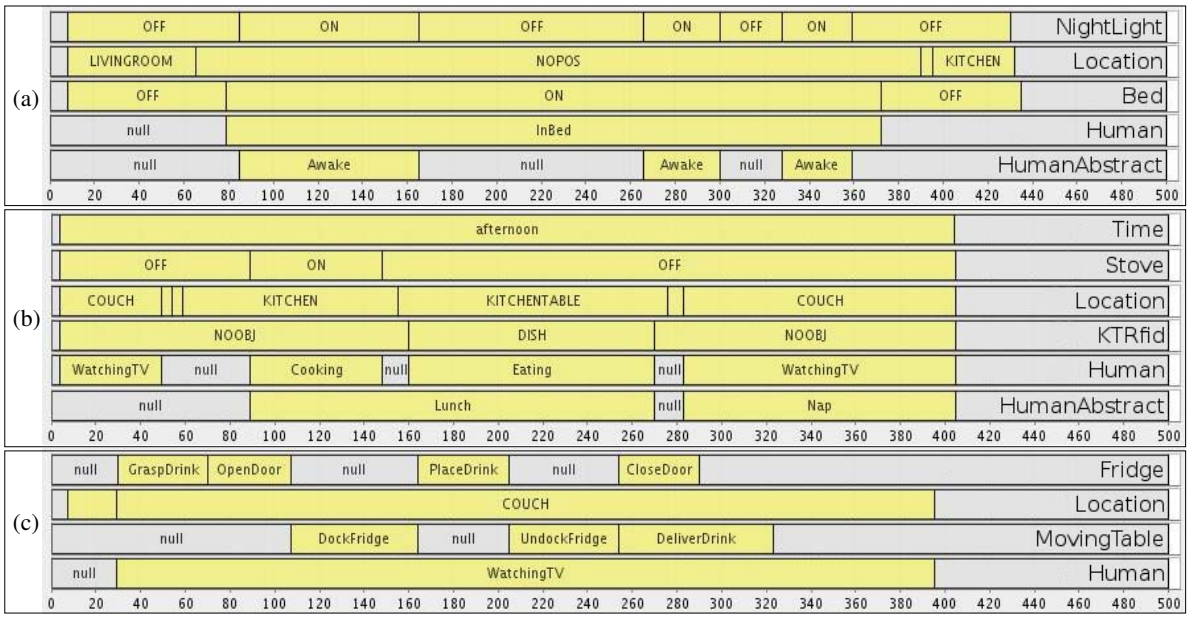

Figure 5: Timelines resulting from the runs performed in our intelligent home using the sleep monitoring (a), afternoon activities (b) and proactive service (c) domains.

One of SAM's current limitations is its relatively simple depth-first search strategy. A more sophisticated re-planning strategy would allow to take into account domains in which more than one synchronization is applicable to support a hypothesis, thus leading to different timelines for the same component. These synchronizations could model, for instance, alternative "explanations" for patterns of sensor readings, or alternative plans that realize different forms of support. Alternative synchronizations on the same values could also enable the synthesis of contingency plans for dealing with actuator execution failures. However, this would inevitably affect the performance of the re-planning procedure, which we have purposefully kept simple in order to maintain re-planning time within the limit of acceptable sampling rates. A first step in the direction of obtaining a performant re-planning procedure is presented in (Ullberg, Loutfi, and Pecora 2009), which details the performance as well as completeness and correctness proofs of SAM's activity recognition functionality.

Acknowledgements. The Authors wish to thank Alessandro Saffiotti for his support as well as the anonymous reviewers for their helpful comments.

References

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

Cesta, A.; Cortellessa, G.; Giuliani, M.; Pecora, F.; Scopelliti, M.; and Tiberio, L. 2007. Caring About the User's View: The Joys and Sorrows of Experiments with People. In *ICAPS07 Workshop on Moving Planning and Scheduling Systems into the Real World*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artif. Intell.* 49(1-3):61–95.

Dousson, C.; Gaborit, P.; and Ghallab, M. 1993. Situation recognition: Representation and algorithms. In *Proc. of 13th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 166–174.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences* 18(2):231–271.

Jakkula, V.; Cook, D.; and Crandall, A. 2007. Temporal pattern discovery for anomaly detection in a smart home. In *Proc. of the 3rd IET Conf. on Intelligent Environments(IE)*, 339–345.

Knight, R.; Rabideau, G.; Chien, S.; Engelhardt, B.; and Sherwood, R. 2001. Casper: Space exploration through continuous planning. *IEEE Intelligent Systems* 16(5):70–75.

Pollack, M.; Brown, L.; Colbry, D.; McCarthy, C.; Orosz, C.; Peintner, B.; Ramakrishnan, S.; and Tsamardinos, I. 2003. Autominder: an intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems* 44(3-4):273–282.

Ullberg, J.; Loutfi, A.; and Pecora, F. 2009. Towards Continuous Activity Monitoring with Temporal Constraints. In *Proc. of the 4th Workshop on Planning and Plan Execution for Real-World Systems at ICAPS09*. (to appear).

Vilain, M.; Kautz, H.; and van Beek, P. 1989. Constraint propagation algorithms for temporal reasoning: A revised report. In Weld, D., and de Kleer, J., eds., *Readings in Qualitative Reasoning about Physical Systems*, 373–381. Morgan Kaufmann.

Wu, J.; Osuntogun, A.; Choudhury, T.; Philipose, M.; and Rehg, J. 2007. A Scalable Approach to Activity Recognition Based on Object Use. In *Proceedings of ICCV 2007. Rio de Janeiro, Brazil*.