



International Master's Thesis

Towards Continuous Activity Monitoring with Temporal Constraints

JONAS ULLBERG
Technology

Towards Continuous Activity Monitoring with Temporal Constraints

*Studies from the Department of Technology
at Örebro University*



Jonas Ullberg

Towards Continuous Activity Monitoring with Temporal Constraints

August 12, 2009

Abstract

Public demand for intelligent services in their home environments can be expected to grow in the near future once the required technology becomes more widely available and mature. Many intelligent home services cannot be provided in a purely reactive fashion though since they require contextual knowledge about the environment and most importantly the activities the residents are engaged in at any given time. This poses a problem since information about a human's behavior is not easily accessible and has to be recognized from aggregated sensor data in most cases. Numerous activity recognition techniques have been studied in the literature. In this thesis we focus on one such technique which takes a temporal reasoning approach to activity recognition, namely recognizing activities by planning for them with a temporal planner. OMPS is an example of such a planner that has been used in previous work to recognize activities of humans in domestic environments. An important requirement for monitoring activities in a real world application is the ability to do so continuously and reliably. Two shortcomings in the previous approach hindered OMPS's capability to meet this requirement, namely maintaining the performance of the activity recognition over long monitoring horizons, and ensuring future temporal consistency of recognized activities. This thesis will define the two problems, detail their solutions, and finally evaluate the modified system with the corresponding changes implemented.

Acknowledgements

I would like to thank my supervisors Federico Pecora and Amy Loutfi for their guidance during this project, and also Lin Wenjie for keeping me good company in the lab room during the writing of this thesis.

Contents

1	Introduction	13
1.1	Domestic activity monitoring	13
1.1.1	Sleeping disorders	14
1.2	Organization of this thesis	15
2	Background	17
2.1	Related work	17
2.2	The OMPS Framework	18
2.2.1	Simple temporal problem	22
2.3	Constraint based activity recognition	26
2.3.1	Sensors	26
2.3.2	Planning for activities	27
2.4	Limitations	28
2.4.1	Consistency	28
	Absolute constraint problem	28
	Relative constraint problem	30
2.4.2	Tractability	31
2.5	The PEIS Ecology middleware	33
2.6	Summary	33
3	Ensuring consistency over time	35
3.1	Absolute consistency check	35
3.1.1	Freezing decisions	38
3.2	Relative consistency check	40
3.3	Summary	44
4	Long term monitoring	45
4.1	Detaching decisions	45
4.2	Planning with detached decisions	46
4.3	Evaluation	50
4.4	Summary	57

5	Implementation and experiments	59
5.1	Developing a testbed	59
5.2	Real world scenario	61
6	Conclusion	67
6.1	Summary	67
6.2	Future Work	68
A	WeekTest.ddl	75
B	Temporal constraints	79

Chapter 1

Introduction

1.1 Domestic activity monitoring

During the last couple of decades a lot of new technology has been brought into common households and it is unlikely that this trend will stop anytime soon. There has been an increasing demand for intelligent home environment appliances such as vacuum cleaning robots and it is realistic to assume that the homes themselves soon will be equipped with a standardized infrastructure to aid in the deployment of robotic devices. Such an infrastructure is likely to include a number of sensors such as cameras for tracking the location of a person or RFID readers that can automatically make inventories about which kinds of foods are available inside the fridge. To maximize their reusability these sensors will most likely also be connected to each other in the household's private computer network.

Such an infrastructure might also be used to monitor the behavior of the inhabitants and use the gathered information to provide different proactive contextualized services for them. Examples of such might include turning off the TV if a person has fallen asleep in front of it or alerting a neighbor or relative if a person known to suffer from epilepsy gets seizures. Another more advanced use of such a system might be warning a person if he is about to get late to work due to the actions he has performed or neglected. All of these applications rely on a monitoring system's ability to provide higher level view of the activities in the environment since they in general cannot be sensed directly. In order to do this a monitoring system needs to be able to provide a rich set of reasoning features which includes reasoning about both the occurrence of events and also their temporal relationship to each other, sometimes during longer periods of time. The monitoring of activities during longer periods of time is especially challenging since it puts high requirements on the reasoning with regards to performance and stability as we will be described in the background chapter of this thesis.

1.1.1 Sleeping disorders

The task of helping medical personnel diagnose sleeping disorders (somniphath) have been chosen as a setting for the activity recognition in this thesis since it is believed to be a good example of how such techniques can have a real world application in the near future (without requiring any actuation of the environment which is not the topic of this thesis). Doctors who are specialized in diagnosing sleeping disorders often provides their patients with a diary in which the patient should note their daily activities during a couple of weeks in order for the doctor to asses the patient's need for further examinations. What is most important in these diaries are the patient's physical activity levels during the course of a day and also their nightly sleeping behavior. There is off course a limit with respect to the level of detail on what can be asked of the patient to note in these diaries, therefore they will often only provide the doctor a very coarse view of their behavior, which might in some cases also be incomplete or give a biased view because the patient has forgotten or neglected to fill in certain activities.

An activity recognition system could be very helpful in this case since it would provide the medical personnel with additional data for study and also relieve the patient of one (albeit small) duty. From an activity recognition perspective this is also an interesting task since the recognition is done during longer periods of time which puts high requirements on the stability and efficiency of the activity recognition system. This poses a significant problem to current techniques based upon temporal reasoning.

1.2 Organization of this thesis

This thesis is organized as follows:

- Chapter 2** Gives an overview of the temporal reasoning technology that has been used in this thesis to recognize everyday activities in intelligent home environments. This chapter also introduces the reader to the two problems that are the topic of this thesis, namely the problem of maintaining temporal consistency and ensuring tractability when recognizing activities over longer periods of time.
- Chapter 3** Proposes a solution to the first problem that is able to identify situations in which activity recognition might fail and also a way to delay the recognition of these activities until they are deemed “safe” so that a consistent hypothesis can be maintained at all times.
- Chapter 4** Proposes a way to speed up the temporal reasoning so that it is able to recognize activities over longer periods of time which solves the second problem.
- Chapter 5** Discusses how the solutions proposed in Chapter 3 and 4 were implemented. It also contains an evaluation of the performance benefits gained by implementing the solution to the second problem along with a test on a real world scenario.
- Chapter 6** Presents conclusions and discusses future work.

Chapter 2

Background

The purpose of this chapter is to give a background to and a motivation for the work done in this thesis. The chapter begins by giving an overview of the OMPS planning framework and how it has been used to recognize daily activities. It then proceeds to describe the two specific problems that are the topic of this thesis and whose solutions are the subjects of further discussions in the rest of the chapters. It also gives a short description about the PEIS Ecology project and the PEIS middleware.

2.1 Related work

Prior approaches to the problem of recognizing human behavior can roughly be categorized by their choice of input data, and processing method. Input data is typically retrieved from sensors placed either on the human [5, 19, 11], in the environment [17, 16], or a combination thereof. Furthermore, the type of the readings can range from boolean values [16] to more complex data types such as images from cameras [2]. Each of these approaches has their own strength and weaknesses, most significantly some of them might be considered invasive or burdensome. Choosing a processing method on the other hand often comes down to a choice between taking a *data-driven* or a *knowledge-driven* approach. A common approach is the data-driven approach which is characterized by the fact that it uses large amounts of data acquired from sensors over time to model human behavior using some kind of probabilistic reasoning technique, for instance, Hidden Markov Models [15] or neural networks [9]. Knowledge-driven approaches on the other hand follow a complementary approach in which patterns of observations are modeled from first principles rather than learned. In these cases the sensor data is explained by hypothesizing about the occurrence of human activities, sometimes through rich temporal representations that model the typical conditions of the environment under which certain activities occur [10]. Data and knowledge driven approaches have complementary strengths and weaknesses. Apart from the obvious drawback with

regards to the labour requirement of the knowledge driven approach, it can be said that the knowledge driven approach is best suited for those cases in which the activities can be easily recognized using common sense while the data driven approach excels at capturing subtle relations but sometimes at the cost of correctness. Furthermore, data driven approaches can relatively easily be adapted to the environment in which they are deployed by the end-user with the help of experience sampling [6], while this might require an substantial effort when using knowledge driven approaches. These are on the other hand better suited for modelling activities only identifiable by people with expert knowledge (e.g. medical knowledge). This thesis takes the latter approach and utilizes the OMPS temporal planning architecture [4] to recognize activities in a domestic environment. Examples of prior similar approaches include [3] and [13] that employ pre-compiled (although highly flexible) schedules as models for human behavior. This differs to the OMPS approach in which the planning process actually instantiates such candidate schedules on-line.

2.2 The OMPS Framework

The Open Multi-component Planner and Scheduler (OMPS) is a software framework written in Java that provides constraint-based planning and scheduling features for complex domains. The OMPS planner has previously been used to build a variety of decision support tools used for tasks ranging from resource scheduling of a mars orbiting satellite to more classical planning scenarios [4].

One of the most fundamental concepts in the OMPS planning framework is the *component* which represents a logical or physical entity relevant to the planning domain. For example, a battery on a satellite is a feasible component in a space mission control domain and a bed is likely to be a component in a sleep related one. Each component in turn has a number of possible states which may vary over time such as remaining power for the battery, or a simple state like “Occupied” for the bed. These states can be “instantiated” on their respective components for given flexible periods of time and such instantiations are called *decisions*. 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 are intervals of admissibility of the start and end times of the decision¹.

The fact that different components have different kinds of behaviors is captured by the concept of component classes in OMPS. The possible values of a satellite’s battery charge, for instance, most likely falls within a bounded range of continuous real numbers and is subject to changes over time depending on which decisions are taken upon the satellite. The battery charge could therefore be modelled as the built-in component class “RenewableResource” which can capture this fact. The bed on the other hand is better modelled as a “StateVari-

¹In this thesis, v is always a single state variable state

able” which is also a built-in component class suitable for modelling entities that can take named states from a fairly small set of possible ones (e.g. “Occupied”, “Unoccupied”, “Made” or “Unmade”)². The OMPS framework has other types of built-in component classes as well and it is also possible to define new custom ones. It is however outside the scope of this thesis to provide a full overview of the OMPS system. For a complete description, the reader is referred to [8].

The planning process makes use of a *domain definition* that defines how decisions taken on one component affect the evolution of other components. An example dependency for an imagined decision “SendData” taken upon an component modelling a satellite’s transmission system might be that the decision should occur DURING a time in which the satellite’s battery is adequately charged. This is also the core intuition behind OMPS; the fact that decisions on certain components may entail the need to assert decisions on other components. And as we will see later, this characteristic of the domain definition will also be leveraged to model the requirements for recognizing human activities from sensor readings.

Dependencies such as these are called *requirements* and are a combination of both *temporal* and *value constraints*, the previously mentioned DURING relationship is an example of a temporal constraint which models the possible temporal relations between the time intervals of two decisions. The temporal constraints in OMPS are bounded variants of those in Allen’s interval logic [1] and Appendix B provides a full overview of each of them. Value constraints on the other hand represent an assertion on the possible values of a decision. A requirement therefore asserts the possible values of a component during a period of time. A set of requirements for a component is called a *synchronization* and models a complete set of dependencies entailed by a decision. Synchronizations are not specified at a per-decision basis however since a decision is an instantiated value of a component, they are specified as preconditions for a given value on a per-component basis. The synchronizations are typically specified in a domain definition file such as the one seen in Figure 2.1 but they can also be specified programmatically.

The example domain definition file in the figure specifies the synchronizations for four different components, for example, a synchronization for a “Dinner” decision on a component named “FamilyActivity” which specifies that it should occur AFTER a decision “Clean(Dining room)” on a “RobotActivity” component. In this case the temporal requirement is AFTER while the value requirement is “Clean(Dining room)” on the “RobotActivity” component³. The AFTER constraint asserts that the decision of having dinner should be-

²A component of the class StateVariable also allows for parametric values, these might be helpful if one wants to model an entity such as a satellite dish, in this case pitch, yaw and roll are feasible decisions and their parameters would be their desired values.

³OMPS also provides support for multiple value possibilities on decisions and in value constraints, these are however not used in this thesis.

```

1 COMPONENT FamilyActivity : SV_FamilyActivity {
2   VALUE Dinner() {
3     AFTER [10,240] RobotActivity Clean_Dining_Room()
4   }
5 };
6
7 COMPONENT RobotActivity : SV_RobotActivity {
8   VALUE Clean_Dining_Room() {
9     EQUALS RobotPosition DiningRoom(),
10    OVERLAPPED-BY [0,INF] RobotBattery Charged()
11  }
12  VALUE Charging() {
13    DURING [0,INF][0,INF] RobotPosition Basement()
14  }
15 };
16
17 Component RobotBattery : SV_RobotBattery {
18   VALUE Charged() {
19     MET-BY RobotActivity Charging()
20   }
21 };
22
23 Component RobotPosition : SV_RobotPosition {
24   % No requirements for "Basement" or "Dining room"
25 };

```

Figure 2.1: A fragment of a domain definition file specifying synchronizations for an example domain.

gin at least 10 but not more than 240 time units after the end time of the “Clean(Dining room)” decision and is an example of the bounded temporal constraints mentioned earlier. The bounded constraints allows us to specify the relative placement of decisions with a greater level of detail then their unbounded counterparts in Allen’s temporal algebra which would only be able to define that the dinner decision can occur at any time after the cleaning decision (which corresponds to the bounded AFTER $[0, \infty]$ ⁴).

Decisions taken upon components and their temporal requirements are maintained in a *decision network* such as the one shown in Figure 2.2. The structure of this decision network complies with the synchronizations specified in the domain definition file example, therefore, a synchronization can be seen as an asserted pattern of occurrence in a decision tree. Some of the decisions in this network also have specified duration constraints which control the minimum and maximum allowable lengths of the decisions, these are not shown in Figure 2.1 but are still defined as a part of the domain definition.

Even though OMPS’s planning process will be discussed later in this chapter, the structure of the decision network and how it relates to the synchronizations defined in the domain are best explained by illustrating how the decision network was built up by planning for a family dinner using the previously defined

⁴OMPS does not implement open intervals and infinity is not truly infinite as will be described in section 2.2.1. Interval bounds for infinity are therefore denoted as closed throughout this thesis.

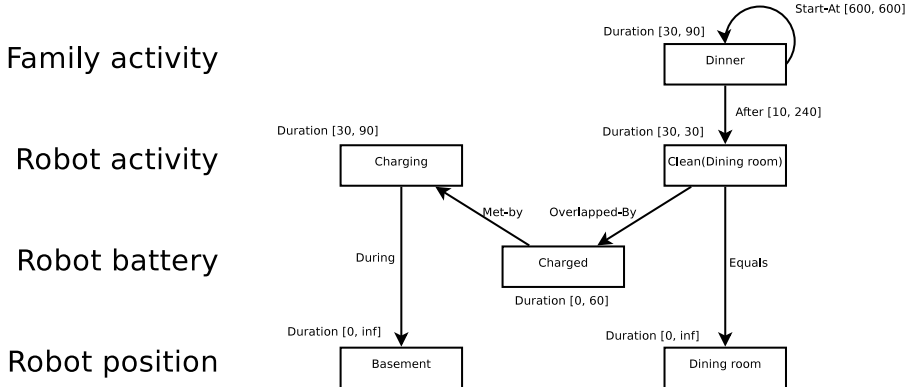


Figure 2.2: A simple decision network that have been built up as a result of planning for a family dinner.

domain synchronizations. The full decision network shows how planning for a dinner that should occur at a specific time instant 600 (which can be interpreted in minutes) first requires the robot to clean the dining room, which entails the robot to be in the dining room during the cleaning. Furthermore, cleaning the dining room requires the robot's batteries to be charged which in turn assesses the need for the robot to charge them down in the basement⁵.

The planning process in OMPS is driven by the goal of supporting *unjustified* decisions. This is done by iteratively performing two different operations which *justifies* unjustified decisions present in the decision network, namely *unification* and *expansion*. When OMPS tries to justify a decision it first tries to *unify* it with another decision already present in the decision network. A unification succeeds if it is possible to add a temporal EQUALS and a value EQUALS constraint between two decisions. If this is not possible the planner instead tries to *expand* the decision by adding its requirements to the decision network as unjustified decisions. This process is iterative since these newly added decisions will be subject to the same justification process as well. The process terminates once there are no unjustified decisions in the network or when all expansion and unification opportunities have been exhausted. In the former case all newly added decisions will be kept in the network while the latter case results in their removal. In those cases where there are no synchroniza-

⁵The scenario was constructed with simplicity in mind and intends to demonstrate OMPS temporal reasoning process without raising questions about value constraints which are not an important topic in this thesis. The assumption about the battery is therefore that it remains charged for 0–60 time units after the robot has charged them (which in reality could be modelled more properly in OMPS)

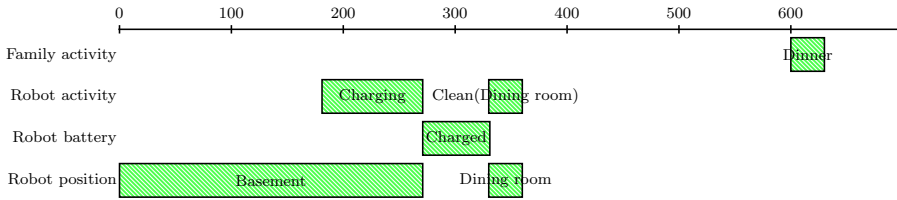


Figure 2.3: Example timeline

tions defined for a decision the decision is directly marked as justified which will be the case for the “leaves” in the decision network. This process might also involve *backtracking* which is a way of choosing a different synchronization (whenever available) to expand in order to justify a decision. In the decision network in Figure 2.2, “Dinner” was initially added as an unjustified decision while the remainder of the decisions present in it was added as a result of expansion, there are no examples of unified decisions in it.

During planning, and at all other times, the decision network is kept consistent through *temporal propagation*. The information acquired during this process can be used to extract a consistent set of timelines for each of the components such as the one seen in Figure 2.3 which shows the earliest start and stop times for all of the decisions in the decision network with respect to the constraints in the decision network. The details of temporal propagation are explained in the following section.

2.2.1 Simple temporal problem

While the decision network gives a high level view of the planning problem its format is not suitable for assessing the temporal consistency of it. Therefore the OMPS planner keeps a second lower-level representation of the temporal problem present in the decision network which it solves as a Simple Temporal Problem (STP). An STP assesses time differences between pairs of *time instants* and has the attractive property that deciding consistency can be done in polynomial time with respect to the number of time instants present [7]. Each decision in the decision network is represented by two time instants at the STP level of OMPS which correspond to the decisions start and stop time while the temporal constraints are translated into one or more *simple distance constraints*. An STP problem can be visualized as a *constraint graph* such as the one in Figure 2.4 in which the nodes represent time instants while the edges are simple distance constraints. This figure is also the STP representation of the decision network shown in Figure 2.2.

By comparing the two figures we can see that the duration requirements in the decision network translates directly into simple distance constraints between the start and end time instants of the decision in the STP. For instance,

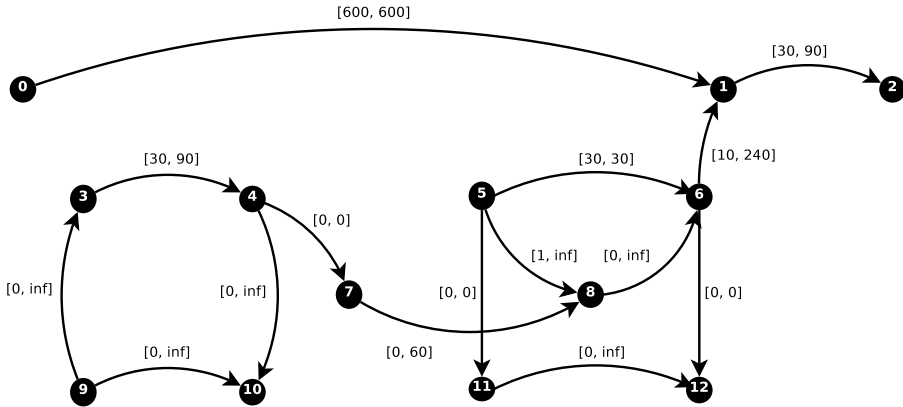


Figure 2.4: A constraint graph that assesses the temporal requirements in the decision network in Figure 2.2.

the decision of having a dinner is represented by node 1 and 2 in the STP and they are bound together by a simple distance constraint that defines that the end time instant (2) of the decision should be at least 30 but no more than 90 time units ahead of the start time instant (1). The start time of the dinner decision is in turn constrained relatively to time instant 0, this time instant does not represent a start or stop time of a decision but defines the start of the planning horizon and acts as an anchor which allows the STP to represent the absolute placement of decisions in time. These are both examples of unary constraints in the decision network, i.e. constraints that only pertain to one decision. The other type of constraints that can be found in the decision network are the binary constraints that define the decisions' relative placement in time. These constraints are sometimes translated into more than one simple distance constraint. The during constraint between the decision of “charging” and “basement”, for example, translates into two simple distance constraints between the pairs of start (3,9) and stop time instants (4,10) of the two decisions. More specifically the translation of the during constraint is built up by one simple distance constraint from the start time instant of the “required” decision to the start time of the “requiring” that asserts that the latter time instant should occur after the former while the constraint between the end time instants follows the opposite logic by asserting that time instant 10 occurs after time instant 4.

The unique characteristics of an STP is that it only allows simple constraints between time instants, meaning that it only admits one single interval per constraint. This contrasts with the general Temporal Constraint Satisfaction Problem (TCSP) which admits multiple intervals per constraint but is NP-hard [7]. When translating the high-level temporal constraints present in the decision

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	600											
1	-600	0	90				-10						
2		-30	0										
3				0	90					0			
4				-30	0			0			inf		
5						0	30		inf			0	
6		240				-30	0		0				0
7					0			0	60				
8						-1	inf	0	0				
9				inf						0	inf		
10					0					0	0		
11						0						0	inf
12							0					0	0

Table 2.1: Unsolved STP

network to the STP level this is however not an issue since each of them have a simple translation. And in those cases where the STP translation results in two or more simple distance constraints for the same pair of time points then the added constraint will be the intersection of the admissible intervals.

The current implementation of OMPS solves the STP Problem by utilizing the Floyd-Warshall algorithm which finds the shortest path between all pairs of nodes in a *distance graph*. By doing this OMPS both asserts the consistency of the decision network and also gets information about each decisions start and stop time instants earliest and latest time of occurrence. A distance graph can be represented in a matrix form such as the one seen in table 2.1 which also correspond to the example STP. The translation from a constraint graph into a distance graph is trivial since they are both very similar in structure and contains the same number of nodes.

A matrix such as the one seen in table 2.1 is easily built from a constraint graph. This can be done directly by separately noting the lower and upper bounds of each interval contained within the constraint graph into a corresponding field in the matrix. Each row in this matrix represents a “from” node and the columns represent a “to” node since each entry represents a upper bound of the distance between the two nodes. As previously mentioned the translation from a constraint graph into a distance graph matrix is easy and is done by noting each interval’s upper bound in the constraint graph into the matrix cell which corresponds to the to–from relationship of the simple distance constraint and then its negated lower bound in the cell that corresponds to the reverse relationship. For instance, the simple distance constraint that enforces the interval [30, 90] between nodes 1 and 2 in the constraint graph will correspond to the entry of 90 in row 1 and cell 2 and -30 in row 2, cell 1. The matrix shown here is fully defined with the constraints from the constraint graph while non-used entries have been left out. These empty cells should be defined as well so that they contain a finite high number which (contrary to the abbreviation) is also the case for those labeled “inf”.

The matrix in table 2.1 can then be solved by applying step 7–11 of Floyd Warshall’s algorithm (included as Algorithm 1) with the results shown in table 2.2. Checking for consistency is trivial once the matrix is fully solved and

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	600	690	560	590	560	590	590	590	560	10000	560	590
1	-600	0	90	-40	-10	-40	-10	-10	-10	-40	9400	-40	-10
2	-630	-30	0	-70	-40	-70	-40	-40	-40	-70	9370	-70	-40
3	-181	419	509	0	90	149	179	90	150	0	9819	149	179
4	-271	329	419	-30	0	59	89	0	60	-30	9729	59	89
5	-330	270	360	0	30	0	30	30	30	0	9670	0	30
6	-360	240	330	-30	0	-30	0	0	0	-30	9640	-30	0
7	-271	329	419	-30	0	59	89	0	60	-30	9729	59	89
8	-331	269	359	-30	0	-1	29	0	0	-30	9669	-1	29
9	9370	9970	10000	9930	9960	9930	9960	9960	9960	0	10000	9930	9960
10	-271	329	419	-30	0	59	89	0	60	-30	0	59	89
11	-330	270	360	0	30	0	30	30	30	0	9670	0	30
12	-360	240	330	-30	0	-30	0	0	0	-30	9640	-30	0

Table 2.2: Solved STP

is done simply by checking so that none of the diagonal elements in the matrix contain a negative value, if this is that case then the STP is inconsistent. On the other hand, if the STP is consistent, then each cell will contain the maximum allowed relative time difference between its corresponding pair of points. Of particular interest are column 0 and row 0 which contain the negated earliest time of occurrence and the latest time of occurrence for the time instants with respect to the anchoring node. The earliest (or latest) times with respect to time instant 0 can then be used to extract one or more timelines which represent one solution to the planning problem such as the one seen in Figure 2.3. In this example inf was set to 10000 and this is what have caused the high numbers in column 10 and row 9 which is due to the fact that the DURING requirement on the basement decision does not constrain the earliest time of its start time instant or the latest time of its end time instant.

Algorithm 1 Floyd-Warshall's algorithm

```

1: for i = 1 to n do
2:   dij ← 0
3: end for
4: for i, j = 1 to n do
5:   dij ← aij
6: end for
7: for k = 1 to n do
8:   for i, j = 1 to n do
9:     dij ← min(dij, dik + dkj)
10:  end for
11: end for

```

The complexity of the simple temporal problem is $O(n^3)$ with respect to the number of nodes in the network (step 7–11). Adding additional constraints or time instants does not require us to repropagate the entire matrix though since it is possible to reuse the old result by only applying step 8–10 which has a complexity of $O(n^2)$ (the removal of one or more constraints still require a full repropagation however).

2.3 Constraint based activity recognition

The previous parts of this chapter have given a general overview of the OMPS framework along with an example of how it handles a simple planning scenario. This section furthers the discussion by describing how it has been used to recognize activities in a home environment and the specific problems that arise when doing so.

2.3.1 Sensors

When OMPS is used for activity recognition the sensors whose readings are used to draw conclusions about different kinds of actions are represented as *non-controllable* components. Non-controllable components are treated differently by OMPS, the difference lying in the fact that the justification process will never expand any decisions on these components, which is the case for the default *controllable* components. Therefore all synchronizations which require a decision on a component which models a sensor, either directly or through a chain of expansions, will fail unless it is possible to unify against a decision that has been taken upon it previously. In the case of sensors this is done over the course of time as new values are sensed in the environment. More specifically, once a new sensor reading is sensed in the environment a new *sensed decision* is added to a non-controllable component with a value which is an interpretation of the sensor reading. A pressure sensor mounted under the mattress of a bed, for instance, might be modelled as StateVariable component named “Bed”, for which the added decision will be an “occupied” decision in those cases in which the pressure is above a certain threshold, and an “unoccupied” otherwise⁶.

This functionality is implemented in a utility library inside the OMPS framework that is constantly polling sensors to detect changes in their (interpreted) readings. If a change occurs, a new decision will be added to the corresponding sensor component with a fixed start time set to the current time of the activity recognition process while the previously sensed decision (if any) will be stopped. The added decision will also have a “DURATION” constraint that specifies the guaranteed bounds on the duration of the decision (typically set to $[0, \infty]$ since its generally unknown). As a last requirement the sensed decision is set to end during (ENDS-DURING) the timespan of a utility-like decision present in the decision network which models the “Future” of the planning horizon. This decision is used to keep the end time of all currently sensed decisions synchronized without requiring any intervention from the polling mech-

⁶This can also be tailored to our reasoning needs, the decision can be left out if we do not have any rule that requires the bed to be in the “unoccupied” state, in which case we only plan for the “occupied” decision. Adding a decision such as “unoccupied” can also be done automatically by OMPS’s timeline completion feature which fills in gaps in a components timeline so that the entire timeline complies with a set of possible value transitions defined at the component level.

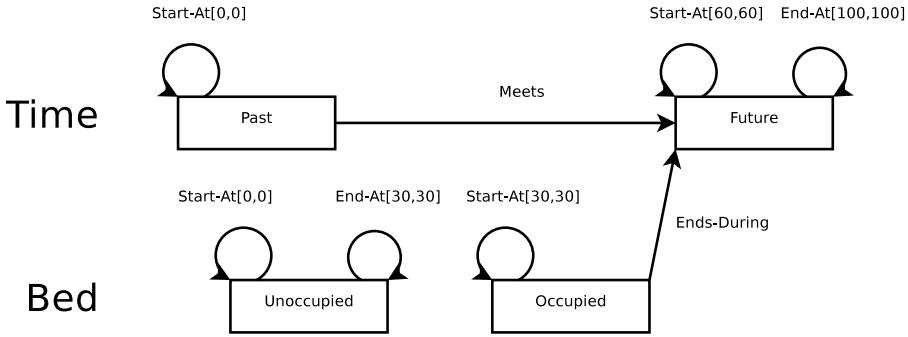


Figure 2.5: Example decision network showing a situation when sensing decisions.

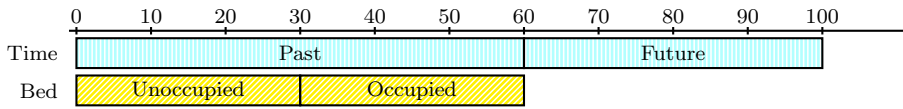


Figure 2.6: An extracted timeline corresponding to earliest start time solution of the decision network found in Figure 2.5

anism. It is accomplished by continuously setting its start time to the current time of the recognition process.

When a sensed decision is stopped on the other hand, the decisions end time is set to the current time of the recognition process (which at this point corresponds to the decisions earliest end time) and the requirement that the decision should end during the “future” is removed.

An example decision network that contains the decisions mentioned earlier can be seen in Figure 2.5. In this figure, the “Unoccupied” decision has been stopped, which can be recognized by the fact that it has a defined end time. The occupied decision however has not stopped yet due to the fact that the interpretation of the sensor reading has remained “Occupied” since time instant 30. It is however prolonged automatically during the course of time as OMPS changes the start-time constraint of the future decision to reflect the real world time. A timeline that corresponds to the decision network is shown in Figure 2.6, it should be noted that the end time of the “Occupied” decision is free to move in time within the interval [60, 100] since the constraint on the decision simply states that it should end during the future, this is a conscious modelling choice that has a benefit that will be discussed in section 3.1.

2.3.2 Planning for activities

Monitoring activities with the OMPS framework is fairly straight forward and is done by repeatedly planning for the activities which should be recognized. A

domain definition when recognizing activities consists of a series of decisions that should be monitored and a series of decisions which specify the possible sensor readings. The synchronizations in the domain are structured so that the decision network which is the result of planning for an activity will always terminate with the unification against one or more sensed decisions, and as previously mentioned, this planning process will never add any decisions to a component that represents a sensor since these are non-controllable. The planning process might however still add decisions on other controllable components. Finally, in order to prevent OMPS from directly unifying a decision that was planned for against a previous decision recognized when planning for an activity, each decision has the additional requirement that it should occur after the latest recognized decision on the same component.

2.4 Limitations

This section of the thesis will discuss the two problems that are the topic of this thesis, namely ensuring the consistency of the decisions network during the course of time and maintaining computational feasibility in the reasoning when recognizing activities during longer periods of time.

2.4.1 Consistency

Even though the way of recognizing activities by planning for them works well in most cases, there are two classes of constructs with regards to sensory input and synchronizations which can lead to inconsistencies in the decision network. The cause of these problems is the simple fact that it is not possible to know the end time of a sensed decision until the physical sensor has changed its reading, in which case the decisions end time is fixed to the current time.

Absolute constraint problem

We will refer to the first class of consistency problem as the *absolute constraint problem* which is characterized by the fact that it occurs when the successful planning of a monitored decision puts absolute bounds upon one or more sensed decision's temporal evolution. The problem is best explained through an example. Consider a domain containing two different components, a human and a bed, where the human models recognized activities and the bed models the sensed state of a bed. The human component has one decision which can be taken upon it, namely "Sleeping", with the requirement that it should have a duration of at least 60 time units and occur "DURING" the timeframe in which the bed is "Occupied". The problem occurs when planning for the activity "Sleeping". When doing this OMPS will first expand the decision of sleeping which yields its requirement of the bed being occupied. This decision is then unified against an "Occupied" decision that is currently being sensed, and

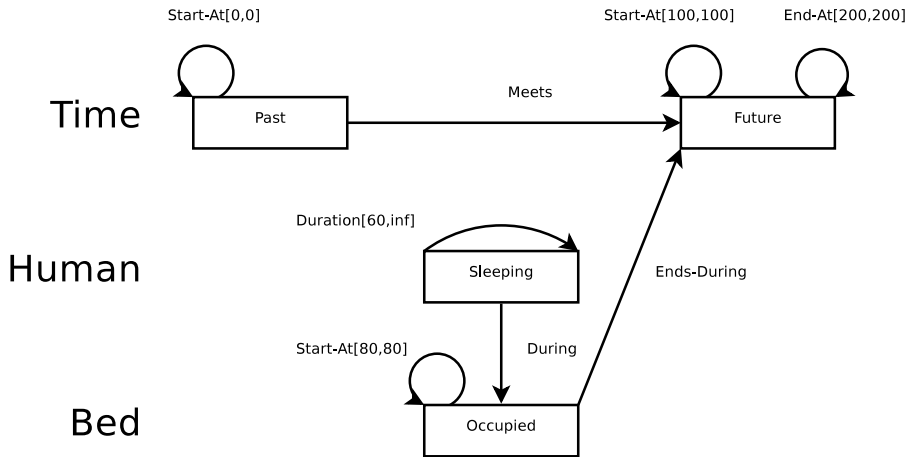


Figure 2.7: Decision network showing the effect of the absolute constraint problem at time instant 100.

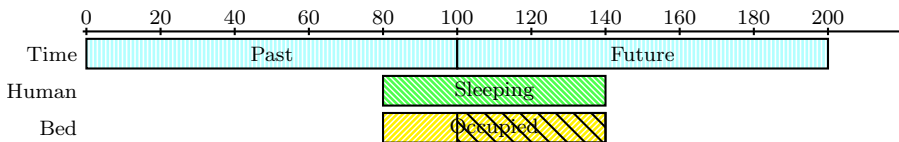


Figure 2.8: Timeline illustrating the absolute constraint problem present in the decision network shown in Figure 2.8. The forced extension of the sensed decision have been striped with diagonal lines in this figure.

therefore allowed to move in time. The unification will succeed which results in the fact that the earliest end time of the sensed “Occupied” decision indirectly gets pushed ahead in time by temporal propagation so that the sensed decision gets a duration of at least 60 time units to satisfy the DURING constraint.

This situation will make the decision network inconsistent in those cases where the real duration of the “Occupied” decision proves to be less than 60 time units long. The propagation failure takes place when the decision is stopped and its end time is set to the current time. This situation is illustrated in Figure 2.7 and 2.8 which shows a decision network and a set of extracted timelines respectively, both at time instant 100 during the monitoring process⁷.

⁷Expanded decisions that have been unified against other decision have been left out from the decision networks in this thesis since they would simply appear as “duplicates” of the decisions which they have unified against. i.e. if included they would have been illustrated as a decision with the same value as their target, bound together with a temporal EQUALS constraint and a value EQUALS constraint.

At first glance one may think that this is simply a modelling problem which could be fixed simply by making the sensed decisions MEET the future while they are being sensed in order to prevent them from being pushed ahead in time in this way. It is certainly true that such a change would allow us to cope with the previously mentioned situation. There is however a duality in this problem since unifications on sensed decision can also constrain the sensed decision's maximum duration instead of their minimum duration (as in the case above)⁸. For instance, a decision modelling a nap with a maximum duration of 60 time units which CONTAINS the sensed "Occupied" decision would be allowed to synchronize with it until the sensed decision's duration has exceeded 60, after which the temporal propagation would fail if the synchronization was taken. A second solution to this problem might be to disallow unifications against sensed decisions which have not stopped yet (or delay the addition of them until they have). This would solve both of the problems mentioned earlier but has the unfortunate drawback that it would prevent the monitoring system to recognize some activities whose synchronizations could be considered to be safe after a certain time (e.g. the sleeping decision can safely be added once the human has been laying in bed for 60 time units).

Relative constraint problem

The second problem regarding temporal consistency during the monitoring process is referred to as the *relative constraint problem*. It is characterized by the fact that it occurs as a result of conflicts between two or more sensed decisions due to the fact that they are both required by one decision that has been planned for, and that the addition of the planned decision puts limitations on their relative evolution in time.

This problem is also best illustrated with an example; consider a scenario involving a human for which we want to recognize one activity, "Sleeping", which synchronizes against an "Occupied" decision on a bed component, and an "Off" decision on a lighting component, so that the sleeping decision is required to occur DURING a time in which the lighting is "Off" and have a time interval which EQUALS the one for the decision that the bed is "Occupied". Planning for the decision "Sleeping" will succeed immediately after the "Occupied" decision has been added to the bed as long as the lighting is "Off" as illustrated in Figure 2.9, which shows the situation at time instant 30.

Unlike the absolute constraint problem, this problem does not put any absolute constraints on the temporal evolution of the end times of the sensed decisions. It does however enforce the fact that the synchronization requires the decision of the bed being "Occupied" to end before or at the same time as the lighting stops being "Off". Thus, the addition of the "Sleeping" decision makes it impossible for the "Occupied" decision to stop before the "Off" de-

⁸i.e. their end time instant's earliest time of occurrence

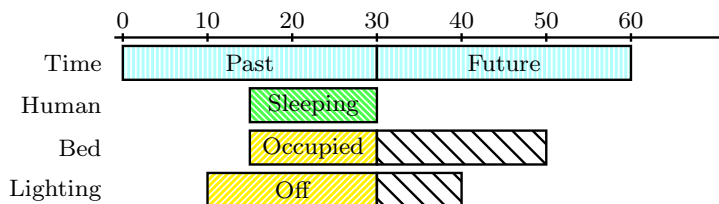


Figure 2.9: Example timeline for three components illustrating the relative constraint problem at time instant 30 along with a possible temporal evolution of two sensed decisions, here indicated by diagonally striped continuations.

cision if the decision network should remain consistent. Naturally, this can not be guaranteed since these two decision’s temporal evolutions are determined by physical events. So if the continuations of the sensed decisions would equal the ones indicated by the striped extensions of the time lines for the bed and the lighting in Figure 2.9, for example, the temporal propagation would fail at time instant 40.

2.4.2 Tractability

The problem of tractability is quite obvious if we consider that propagating one constraint in the STP is an operation with a complexity of $O(n^2)$, where n is the number of time instants in the STP. Or even worse, a $O(n^3)$ operation when removing constraints since this require a full repropagation of the entire STP. This is not an issue for smaller planning and recognition tasks involving few decisions in the decision network, but as the number of decisions grow, maintaining correct bounds on decisions while continuously monitoring may lead to slower updates of the recognition process.

When planning for a decision in order to recognize it, as described before; the decision is added to the decision network and then expanded or unified, in which case the expansion results in the addition of new decisions subject to the same procedure. The series of expansions always ends with unifications against other decisions, and to be able to ascertain if a given combination of unification targets satisfy the constraints in the decision network, each combination has to be tried separately.

The case in previously mentioned scenario in which the “Sleeping” decision for the human requires the lighting to be “Off” and occur during a time in which the bed is “Occupied” is shown in Figure 2.10. This figure contains two decisions of the bed being “Occupied” and three decisions of the lighting being “Off”. The decisions in the dotted boxes in this figure represents the decisions that were created during the expansion of the “Sleeping” decision’s synchronization. In order for OMPS to test if there is a combination that satisfy the constraints in the decision network during unification (which is done

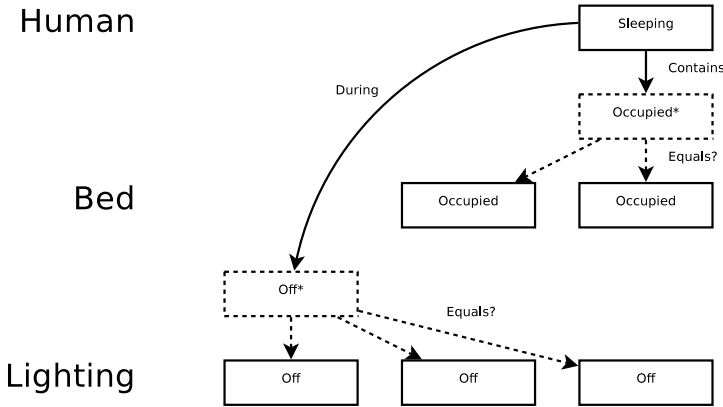


Figure 2.10: Decision network showing possible unifications. Decisions used for unifications has been marked with asterisks.

by adding a temporal EQUALS (and a value EQUALS) constraint between the “real” decisions and the ones used for unification), OMPS tries all opportunities, which in this case requires $2 * 3 = 6$ operations (i.e. the Cartesian product of the number of unification opportunities for each unjustified decision). Each of these operations consists of one unification against an “Occupied” decisions, and one unification against an “Off” decision. In turn, each such consistency test require the propagation of the newly added EQUALS constraints.

The problem of tractability therefore consists of two parts when planning for a decision. The first part requires us to test a number of combinations of targets for a synchronization, and the complexity of doing this depends on the number of requirements and unification opportunities. The second part consists of the propagation of the STP whose complexity depends on the number of time instants in it. However each specific combination that is tried in the first part requires the propagation of the STP so the problems are interleaved.

OMPS does however implement some filtering on the combinations of unifications to try. The effectiveness of this is however very dependent on the type of domain since it, amongst other, relies on how often decisions are taken (in the case of activity recognition), and more importantly, the type and number of temporal requirements in the synchronizations.

It can be expected that the number of decisions in the decision network grows quickly when performing 24 h / day monitoring of activities if there are a lot of events taking place in the environment. The problem of recognizing activities therefore quickly turns infeasible, both due to the size of the decision network and the number of unifications that has to be tried.

2.5 The PEIS Ecology middleware

We finish this chapter by giving a short description of the *PEIS Ecology middleware* [14]. The PEIS Ecology middleware is a middleware for ubiquitous robotics that has been used in conjunction with OMPS in this thesis to allow for the retrieval of sensed values from sensors placed in a domestic environment. It is developed as a part of the *PEIS Ecology project* which is a collaborative research project between the centre for Applied Autonomous Sensor Systems (AASS) in Sweden and the Electronics and Telecommunications Research Institute (ETRI) in Korea. The goal of the project is to provide an infrastructure for ubiquitous robotics in smart robotic environments that allows for the cooperation of many simple networked robotic devices in the service of people. This is achieved through the PEIS middleware that provides a shared communications platform for such devices.

In the PEIS middleware, devices communicate with each other by reading and writing *tuples* in a distributed *tuplespace*. A tuple is a key/value pair, in which the key is a string that identifies the meaning of the value which in turn can be of any type (e.g. a string or binary data). Therefore, the PEIS middleware can be said to provide a distributed memory model for robotics communications. Furthermore, each tuple has an owner which is the device which keeps the defining instance of the tuple, the tuples are then shadowed on devices that *subscribe* to them, and changes done to the defining instance are distributed to all subscribing devices. This is advantageous from a programming point of view since network latency is not an issue when reading tuples.

Even though the current implementation of the activity recognition code in OMPS retrieves its sensory input from the tuplespace in a PEIS Ecology, it could easily be adapted to be used in conjunction with other systems as well.

2.6 Summary

This chapter has given an overview of the previous work done in the field of activity recognition using the OMPS planner and has described the problem that are the topic of this thesis, namely the problem of maintaining temporal consistency and ensuring tractability when recognizing activities over longer periods of time. We also concluded with a description of the middleware used to retrieve sensor readings. The concepts summarized in this chapter are relevant to the main contributions, detailed in the following chapters. Specifically Chapter 3 will describe how the problem of ensuring future temporal consistency of the decision network was solved and Chapter 4 describes how long term monitoring was enabled with some performance improvements. Finally Chapter 5 describes how the system was implemented within the PEIS Ecology.

Chapter 3

Ensuring consistency over time

As previously mentioned in Chapter 2, there were two problems that could lead to inconsistencies in the decision network when planning for activities. The first problem was characterized by the fact that it occurred when the successful planning for a monitored decision put an absolute limit on a sensors evolution in time. The second problem was similar but instead occurred when the planning for a decision put limitations on two or more sensors relative evolution in time. This chapter will discuss two methods that were developed to identify these dangerous situations and delay the decision making until the decision can be taken without risking future inconsistencies.

3.1 Absolute consistency check

The absolute consistency check identifies the first consistency problem which occurs when the successful planning of a monitored decision constrains the future evolution of a sensed decision. Intuitively, the solution developed to identify this situation consists of taking a snapshot of the flexibility of all time instants in the STP before planning for a decision. These flexibilities are then compared to their corresponding ones found in the resulting STP after planning. If it is the case that the flexibility of a monitored decision gets reduced during the planning, the decision planned for gets removed from the decision network.

Specifically, the pseudo code for doing this is included as Algorithm 2 where the underlying STP can accommodate at most NUM_TPS time instants. Here, line 2 copies each time instant's flexibility, and line 7 compares these to the new ones after planning. In this algorithm, d_{0i} and d_{i0} are the earliest and latest times of occurrence of time instant i (as in Algorithm 1), here used to compute each time instant's flexibility f_i . In addition, we assume that entries for unused time instants in the distance matrix are set to zero. E.g. if a time instant j is unused then $d_{0j} + d_{j0} = 0$. If it is the case that a decision is successfully planned for (line 5), and that this constrains another time instant previously

found in the STP (like the end time of a sensed decision as in Figure 2.7), then the decision network gets restored to its previous state, which is not shown here but indicated at line 8. This is also the reason why the sensed decisions end time are kept flexible until they are being stopped, if this would not have been the case these situations would have been impossible to detect at the STP level.

Algorithm 2 Absolute consistency check algorithm

```

1: for i = 1 to NUM_TPS do
2:    $f_i \leftarrow (d_{0i} + d_{i0})$ 
3: end for
4:
5: if Plan() then
6:   for i = 1 to NUM_TPS do
7:     if  $(d_{0i} + d_{i0}) < f_i$  then
8:       return false
9:     end if
10:   end for
11: end if
12:
13: return true

```

In order to be generic, the absolute consistency check does not make any difference between sensed decisions and other ones, meaning that it does not allow any time instant in the STP to be constrained, without regards to if they belong to a sensed decision or not. Even though a less restrictive check could allow time instants which does not belong to a sensed decision to be constrained and still guarantee future temporal consistency, this was not seen as a desired feature since it is an indication on the possibility of a “race condition” in the planning process, so that the successful planning for one decision would disallow other decisions to be taken.

An example of a race condition is shown in Figure 3.1 which contains two sensed decisions taken upon a “Bed” and a “Lighting” component, and one monitored decision taken upon a “Human” component. In this case the decision of “Resting” taken upon the human is flexible to a certain extent, so that it can start within the interval [10, 40] and end within [60, 90], this effectively gives the decision of “Resting” a minimum duration of 20, or a maximum of 80. The issue in this case is that both of the decisions on “Rest-State”, “Nap” and “Sleep”, could synchronize with the decision of “Resting”, however they cannot do this at the same time since adding one would prevent the other one from being taken. This is due to the fact that both of their synchronizations indirectly constrains the “Resting” decision’s duration with an EQUALS constraint. The resulting timelines for the two possible synchronizations can be seen in Figure 3.2 and Figure 3.3.

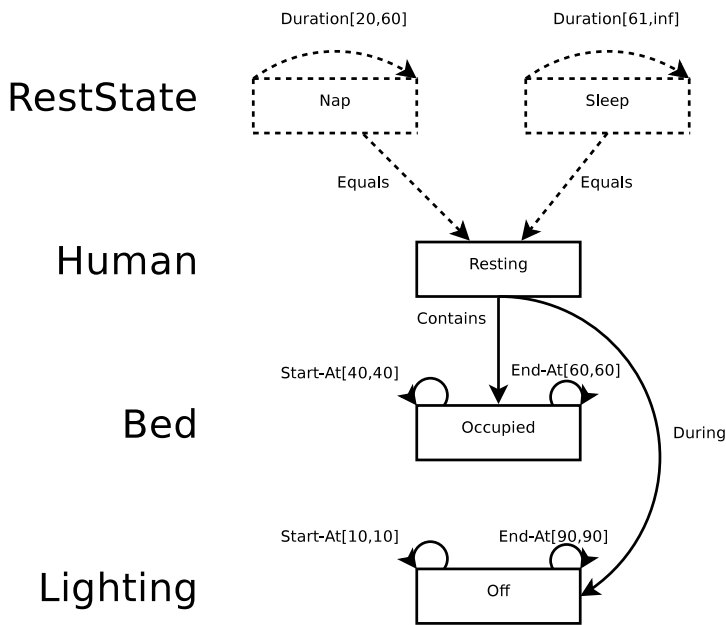


Figure 3.1: Decision network showing a situation in which OMPS has to choose between two possible synchronizations.

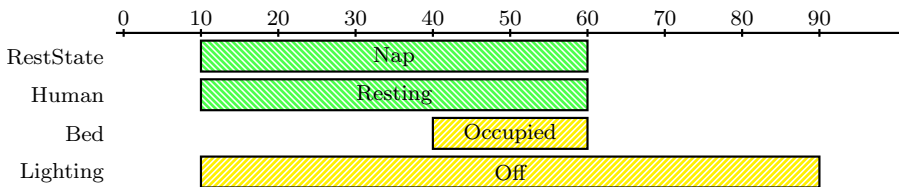


Figure 3.2: Extracted timeline for the decision network in Figure 3.1 after synchronizing the “Nap” decision.

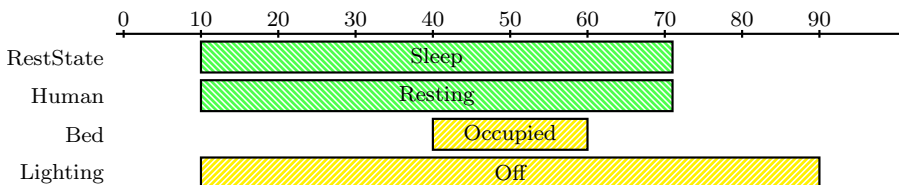


Figure 3.3: Extracted timeline for the decision network in Figure 3.1 after synchronizing the “Sleep” decision.

Strictly disallowing all synchronizations which constrains any time instant previously found in the STP during planning does lead to some problems though since we in general want to be able to specify synchronizations in the domain that assesses the interval of admissibility of the time instants in the STP with respect to their earliest and latest start time. The solution to this was to introduce the concept of *frozen decisions* and implement a way to *freeze* decisions which allows them to be used as targets of synchronizations that would otherwise have been considered hazardous by the absolute consistency check.

3.1.1 Freezing decisions

A decision is considered to be frozen if the upper and lower limits of its start and end time are set to the same value, meaning that the decision is fixed in time. Decisions that are being sensed are considered to be frozen once the interpretation of the physical sensor's reading has changed and they have gotten their end time fixed to the current time. The decisions that are created as a result of planning for activities does not necessarily have this property however since they can sometimes be freely moved in time even after all of their underlying sensed decisions have stopped (as in Figure 3.1). To account for this the monitored decisions are being frozen by fixing their start and end times to their earliest possible values by adding a corresponding requirement to the decision network. This is also the solution shown in the extracted timelines in this thesis, i.e. an earliest start-time solution.

A decision cannot be frozen at any time however since the act of freezing would then be able to constrain the temporal evolution of an underlying sensor. The requirement to be able to freeze a decision is therefore that all of its required decisions are frozen first. This works since the decision network created when planning for activities has a strict hierarchy, i.e. there are no dependency loops in the decision network when planning for activities, such as A requires B and B requires A, in which case it would not be possible to freeze them without requiring the use of some sort of mark-and-sweep algorithm. Finally, at the same time as a decision is frozen all of its requirements and expansions that are only used for unifications are removed to reduce the size of the decision network slightly, this is illustrated in Figure 3.4.

Freezing decisions certainly prevents us from unifying against decisions in those cases in which the earliest start time solution is not adequate to satisfy the requirements of a synchronization, and for which the original flexible decision could have been adapted in favor of the synchronization. This does not necessarily have to be perceived as a drawback however since one way of looking at the issue could be that the flexibility should be in the requirements of a synchronization rather than in the decisions which they rely upon. This is preferable from a modelling perspective since synchronizations are likely to be defined without regards to any flexibility in the underlying decisions which they require, and even if such synchronizations are defined, they tend to create prob-

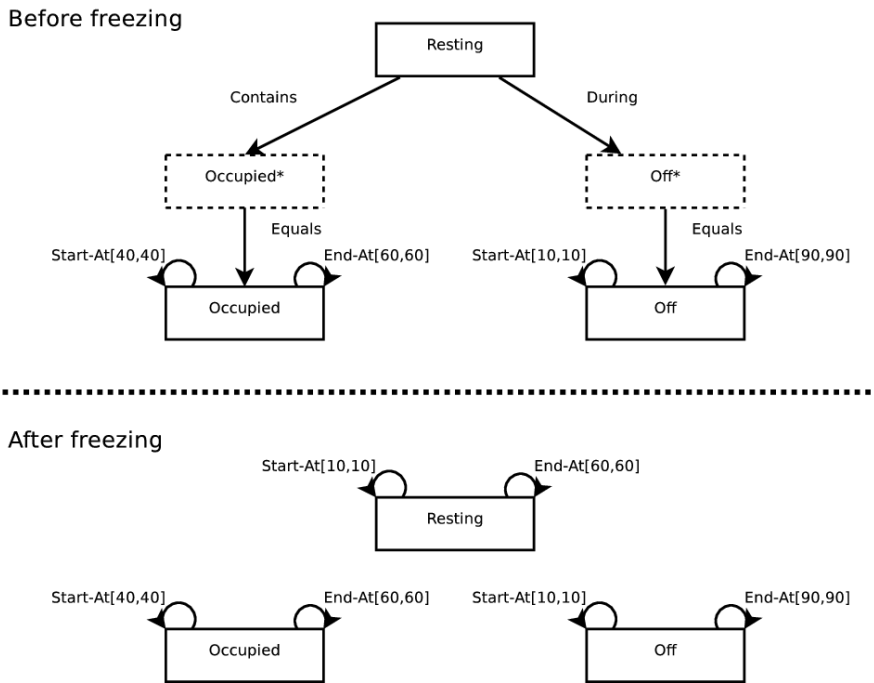


Figure 3.4: Two views of a decision network; before and after the freezing of the “Resting” decision.

lems such as the “race condition” described earlier. This is not an issue when freezing since it gives the decisions that was planned for one specific timeframe that only depends on the targets of their synchronizations.

It should be noted however that the absolute consistency check would not have required the decisions of “Resting” to be frozen before taking the decision of having a “Nap” since the addition of having a nap does not constrain the earliest and latest start time of the “Resting” decision. A partial proof of this is the fact that the extracted earliest start times shown in the timeline in Figure 3.2 for the “Resting” decision would be equal to an extracted timeline in which neither “Nap” or “Sleep” had been synchronized. The additional fact that makes this possible is that the latest end times of the resting decision start and stop time instants are not affected by the addition of “Nap” since it could begin at time instant 40 and end at 90.

To summarize we can state that synchronizations that are satisfied with both the earliest start/end time solution and the latest start/end time solution of their required decisions can be taken directly without first freezing their required decisions. If however a synchronization is satisfied with the earliest time solution but not the latest, then it can be taken first when its required decisions have been frozen (as would be the case for “Nap” if “Occupied” began at time instant 20, since the addition of “Nap” would limit the latest end time of the “Resting” decision to 80). The remainder, i.e. those that are not satisfied with the earliest start time solution, will not be taken at all.

3.2 Relative consistency check

We refer to the second check that is done to assert the consistency of the network as the *relative consistency check*. This check makes sure that the planning of a monitored decision does not put any relative temporal limits on the bounds of its required decisions (as described in section 2.4.1). In a way similar to that of the absolute consistency check, this check also requires a snapshot to be taken before planning. In this case however the snapshot is taken on the decisions and requirements in the decision network and not on the earliest and latest times of the time instants in the STP (as in the case with the absolute consistency check). The snapshot is used after the successful planning of a monitored decision to assess the difference between the old and the new network and thereby determine which decisions was added during the planning process. This information is then used to build a small STP that only includes the STP translation of the newly added requirements and their required time instants. Therefore, the resulting STP will contain a copy of all of the simple temporal constraints and time instants that were added during the planning and also the old time instants that were not added during the planning but referenced by the newly added constraints. As a final step, each of the old time instants present in the new STP are set to their upper and lower limits one by one to make sure

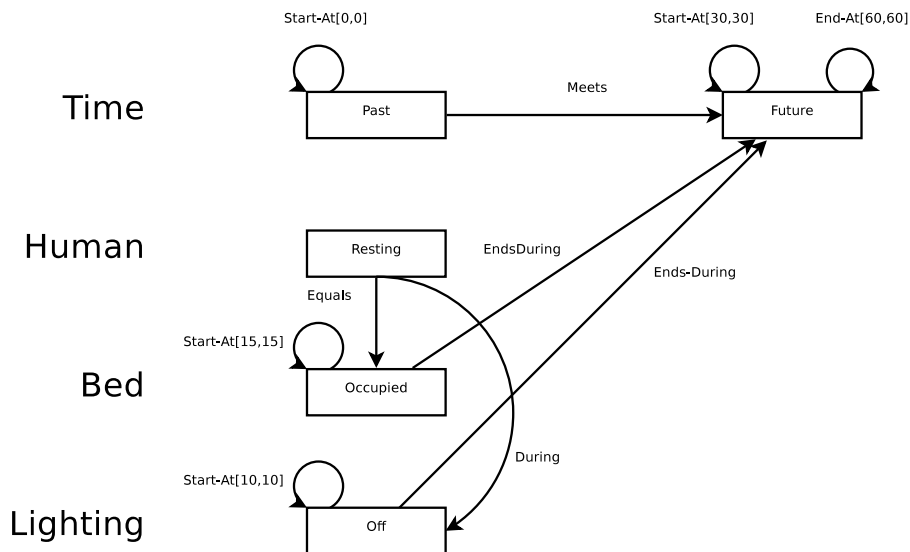


Figure 3.5: Decision network showing a situation that has a relative constraint problem.

that a change in their time of occurrence does not affect the temporal bounds of any time instant that were not added during the planning.

The exact details of this is however best explained through an example. Figure 3.5 shows a decision network that models the example of the relative constraint problem described in section 2.4.1. In this figure the decision of “Resting” has been successfully added by the planning process, there is however a risk of future inconsistencies. If the decision “Off” would stop before the decision “Occupied” the decisions network would become inconsistent due to the fact that the decision of “Resting” would not be able to occur in a timeframe which EQUALS that of the “Occupied” decision and at the same time occur DURING the time in which the lighting is off. If the relative consistency check is done at this point in time, which it is, it would compare the old snapshot of the decision network taken before the planning to the one after planning. This would reveal that the decision “Resting” that was planned for had been added to the decision network along with two decisions used for unification (which are however not shown in the figure). The information about the added decisions will then be used to build a smaller STP such as the one seen in Figure 3.6. This STP includes the STP translation of all the requirements that was added to the decision network and also the time instants that corresponds to the decision that was planned for (1 & 2), the decisions created for unification (3 & 4, 7 & 8, marked with an asterisk), and finally the ones that belong to the target decisions of the two unifications (5 & 6, 9 & 10).

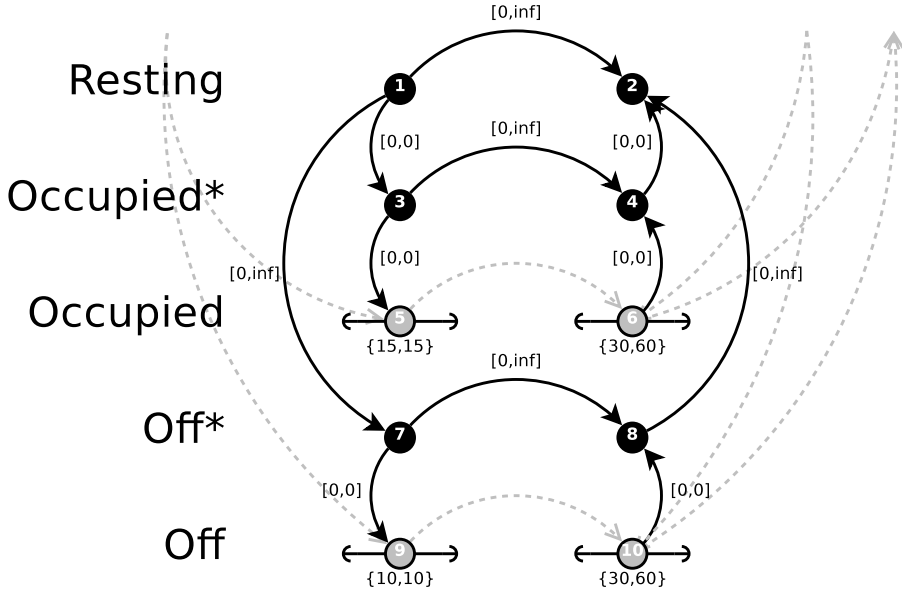


Figure 3.6: Partial constraint graph illustrating an STP used during a relative consistency check.

It would have been possible to simplify this figure by removing the time instants which belongs to the two decisions used for unification since they are constrained to the time instants of the required decision with admissible intervals of $[0, 0]$ (which effectively makes them equal from a temporal reasoning perspective). They are however included both for the sake of correctness, and also to illustrate that the partial STP can have a more complex structure than it would if they had been left out. Another case in which the partial STP would have a more complex structure is in those cases in which a decision that was planned for does not synchronize against sensors directly after an expansion but rather through a (sometimes branching) chain of expansions which ends with unifications against sensed decisions. For example, planning directly for either the “Nap” or “Sleep” decision in Figure 3.1 would add the decision of “Resting” if it were not previously there which would result in a partial decision network with more time instants.

The relative consistency check is performed on a STP such as this one and is done by separately setting all old time instants’ time of occurrence to their earliest and latest times while verifying that these changes does not limit any other old time instant’s timeframe. The time instants intervals of admissibility have been denoted with braces in this figure and their values corresponds to those in the original STP. In this particular figure the risk of a future inconsistency will

be found when setting time instant 6's time of occurrence to its upper bound of 60 which propagates to constrain time instant 10's earliest time to 60 as well. This corresponds to the situation in which the bed is "Occupied" until time instant 60, which indirectly requires the lighting to be "Off" until time instant 60 as well, which of course cannot be guaranteed because "Off" is a sensed value.

Algorithm 3 shows the details on how each time instant is being fixed to its earliest and latest time of occurrence (line 2 & 10), and how this change is assessed to not limit the bounds of any other time instant (line 4 & 12). This algorithm assumes that planning has already been done and that the ids of the old time instants are stored in `oldTIs`, while `oldETs` and `oldLTs` contains the time instant's previous earliest and latest times of occurrence. The functions used are "Constrain" which constrains two time instants relative time of occurrence (in this case relative to ti_0 which sets the time instants absolute time of occurrence), "RemoveConstraint" which removes the added constraint, and "ET" and "LT" which retrieves the current earliest and latest times of occurrence for a time instant from the `miniSTP`.

Algorithm 3 Relative consistency check algorithm

```

1: for  $ti_a$  in oldTIs do
2:   Constrain(miniSTP,  $ti_0$ ,  $ti_a$ , [oldETs $_{ti_a}$ , oldETs $_{ti_a}$ ])
3:   for  $ti_b$  in (oldTIs \ { $ti_a$ }) do
4:     if LT(miniSTP,  $ti_b$ ) < oldLTs $_{ti_b}$  then
5:       return false
6:     end if
7:   end for
8:   RemoveConstraint(miniSTP,  $ti_0$ ,  $ti_a$ , [oldETs $_{ti_a}$ , oldETs $_{ti_a}$ ])
9:
10:  Constrain(miniSTP,  $ti_0$ ,  $ti_a$ , [oldLTs $_{ti_a}$ , oldLTs $_{ti_a}$ ])
11:  for  $ti_b$  in (oldTIs \ { $ti_a$ }) do
12:    if ET(miniSTP,  $ti_b$ ) > oldETs $_{ti_b}$  then
13:      return false
14:    end if
15:  end for
16:  RemoveConstraint(miniSTP,  $ti_0$ ,  $ti_a$ , [oldLTs $_{ti_a}$ , oldLTs $_{ti_a}$ ])
17: end for

```

This check does not make any difference between time instants belonging to sensed decisions or monitored decisions and this would not have been possible either. This is due to the fact that the temporal evolution of the sensed decisions can propagate through many monitored decisions which effectively makes the reasoning about monitored decisions as error prone as reasoning about sensed decisions. In other words, it would not have been possible to only test the time

instants which belongs to sensed decisions since it is possible that the planning for a decision resulted in a unification against one or more monitored decisions whose temporal evolution in turn are determined by sensed ones. One drawback with this approach however is that it does not take into account that the interval of admissibility of some time instants are not affected by the temporal evolution of sensed decisions. An example of this can be created from Figure 3.1. If it is the case in this figure that the “Nap” decision has been successfully planned for, the relative consistency check would have tried to move the start and stop time instants of the “Resting” decision within its original interval of admissibility, [10, 40] and [60, 90] respectively, which would have caused a conflict when setting the start time instant of the “Resting” decision to its earliest time of 10 since it would limit the latest time of occurrence of the same decision’s end time instant to 70. This situation will be resolved once the decision of “Resting” has been frozen in time though.

3.3 Summary

Used together, the relative and absolute consistency checks are able to ascertain that the decision network remains consistent at all times when recognizing activities. This assumption has been tested by planning for decisions with synchronizations known to be hazardous on random sensory input (e.g. as in Benchmark #7 in section 4.3). Due to the fact that a synchronization can theoretically consist of any number of temporal requirements, which in turn can be of different types, it is impossible to give examples on how OMPS behaves in all situations. We can however summarize this chapter by stating that simple synchronizations such as $A \text{ EQUALS } B$, or $A \text{ CONTAINS } B$, never fails unless A has a duration constraint imposed upon it. Furthermore, synchronizations taken exclusively against sensors might be delayed in those cases in which the decision that was planned for has a duration constraint imposed upon it, and/or if the synchronization is done against more than one sensed decision. In these cases the planning will however not be delayed more than necessary and the decision will be taken as soon as possible. When unifying against one or more monitored decisions the situation is however a bit more complex, in these cases the planning might be delayed until some of its required decisions have been frozen. Decisions are frozen as early as possible though, so unless the monitored decisions have a long duration this should not be so much of an issue.

Chapter 4

Long term monitoring

The previous chapter have discussed two methods that makes sure that the decision network is kept consistent at all times when monitoring activities. Both of these solutions however required decisions to be frozen in time which effectively removed all of their temporal flexibility. Therefore, the act of freezing could be considered to be a way to trade some of the reasoning power inherent in the flexibility of the monitored decisions for the ability to guarantee the future temporal consistency of the decision network. This did indeed limit the reasoning power of OMPS somewhat, it was however concluded that this would rarely be a practice issue. This chapter will explain how the non-flexible property of the frozen decisions can be exploited to speed up the planning significantly by removing the frozen decisions from the decision network without making them unavailable to the temporal reasoning. It also contains an evaluation of the performance increase that is the result of these changes.

4.1 Detaching decisions

The process of *detaching decisions* consists of removing frozen decisions from the decision network and storing each decision's start and stop time along with its value in a simple data structure outside the scope of OMPS ordinary planning system. This increases the performance of the reasoning since it reduces the number of time instants that has to be propagated in the STP. A decision can be detached if it is frozen and there is no temporal requirements between it and another decision in the decision network. This is a property that all decisions will get as decisions in the decision network are frozen due to the removal of underlying requirements in the freezing process, as in the lower part of Figure 3.4. Thus, its removal will not affect the earliest and latest end times of the other decisions in the decision network.

The current implementation of the detaching process keeps a user-specified minimum number of decisions present in the decision network at all times which will not get detached until new decisions are taken upon the same com-

ponent, and the reason for doing this will be discussed in the next section. It can however be stated here that the reason for a keeping a fixed number of decisions active instead of decisions within a recent time interval is that the former approach is invariant to the time scale used and this was considered to be a beneficial property. Decisions are then sorted by their end times when detaching is done so that a decision that ends at an earlier point in time will get detached before one that ends at a later. In the remainder of this thesis we will refer to the class of decisions that have been detached as *detached decisions* while those present in the decision network will be referred to as *active decisions* (disregarding if they are frozen or not).

4.2 Planning with detached decisions

OMPS's ordinary planning process is as mentioned in Chapter 2 driven by the goal of justifying decisions, either by expanding them, which results in the addition of new decisions, or by unifying them against other decisions in the decision network. As the case is with detached decisions it is of course not possible to unify a decision against them since they are not present in the decision network. This was however made possible in OMPS through the addition of another level of reasoning that enables detached decisions to be used as targets of requirements in those cases in which the ordinary planning system fails to unify one or more decisions against the ones found in the decision network.

While OMPS's ordinary justification process justifies a decision through unification by adding a temporal EQUALS and value EQUALS constraint between an unjustified decision and a justified one, the unification against a detached decision is done a bit differently. Instead of adding a value EQUALS constraint between the unjustified decision and the detached decision, a simple lookup is done in a list of detached decisions for the target component. This is done in order to find suitable candidates that have the same value as the decision that should be unified. For each of these suitable candidates, OMPS then tries to set the start and stop times of the decisions which would have been unified with other decisions in the decision network in the old system to the start and stop times of their corresponding detached decisions.

In this way there is no difference from a planning point of view between unifying against an active decision and a detached one. An example of this can be seen in Figure 4.1. The left part of this figure shows how the decision of "Resting" has been expanded and the resulting decisions unified against two active decisions, "Occupied" and "Off". The right part of the same figure shows the same unifications but in this case the expanded decision "Occupied" has been unified against a detached decision instead.

The total number of decisions in the decision network will remain fairly the same during the course of time since the current implementation of the detaching procedure keeps a fixed number of decisions for each component active in the decision network while detaching the rest of them. Thus, the propagation

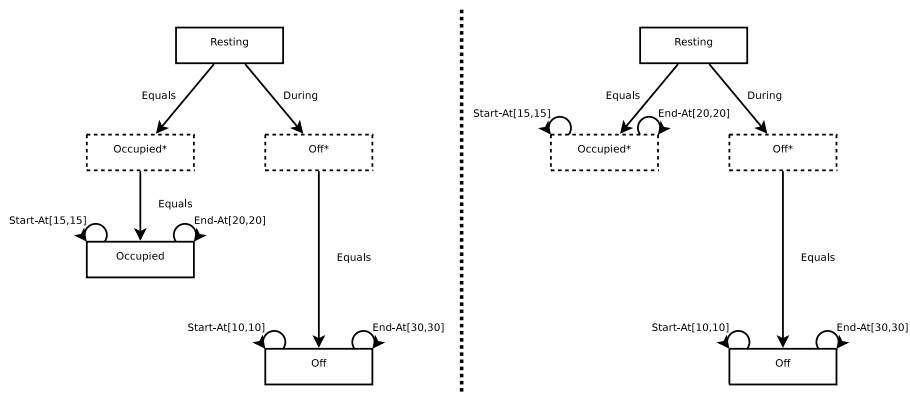


Figure 4.1: Two decision networks showing the difference between unifying against an active decision (in the decision network to the left) and a detached one (in the decision network to the right).

cost in the STP will not increase during the course of time. When it comes to the number of unification combinations that needs to be tried there is however no performance increase in a naive approach, which tries all combinations of active and detached decision, since it will try as many combinations as it would if no decisions had been detached. In this way the basic cost of propagating the STP will remain the same during the course of time but the number of combinations that needs to be tried will grow in a non-linear fashion (depending on the synchronizations in the domain).

Notice, though, since planning is done continuously for the decisions that should be recognized, one way of looking at the recognition process is that it is waiting for a complete set of decisions (sensed or monitored) that are able to satisfy the requirements of some decision's synchronization (i.e. all decisions that could have been planned for have been planned for). The only way in which such a set can be created is through the addition of a new decision, and since new decisions “arrive” as active decisions, it will always be the case that a decision that is successfully planned for uses at least one active decision as a target of its synchronization. This makes it possible to prune the search tree accordingly by putting a requirement on the unification against detached decisions that states that at least one of the decisions in a synchronization should be active. Decisions that are active and non-frozen can however fail to satisfy a requirement in a synchronizations before they get frozen. Therefore, to give the monitoring system some time to use these, a number of recent decisions are always kept active in the decision network for each component as men-

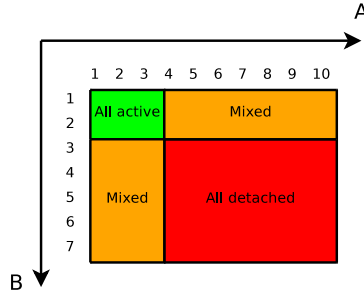


Figure 4.2: An illustration of the different types of combinations that can be chosen when unifying with regards to detached and active decisions. In this case the synchronization requires two values, A and B, a higher order synchronization would have been represented by a partitioned hyperrectangle.

tioned in section 4.1¹. The completeness of the planning is guaranteed as long as the monitored decisions are planned for as often as their required decisions are sensed (in the worst case when only one decision is kept active on each component).

The added requirement that states that at least one of the target decisions of a unification should be active increases the performance during long term monitoring since it can be expected that the bulk of the decisions will be detached. For example, consider a synchronization that requires two decisions A and B, and that there are 7 detached and 3 active decisions with the value A, and 5 detached and 2 active decisions with the value B. If the requirement is that at least one of the decisions should be active only $(3 * 2) + (3 * 5) + (2 * 7) = 35$ combinations needs to be tried while a naive approach would in this case have to try $(7 + 3) * (5 + 2) = 70$ combinations. In the latter case, when a synchronization has two requirements, the number of combinations that needs to be tried will instead grow linearly over time as decisions are added. This is also illustrated in Figure 4.2 in which the detached unification does not try combinations in the “All detached” group.

More specifically, a normal synchronization will require

$$\prod_{r=1}^n (D_r + A_r)$$

combinations to be tried, where n is the number of requirements for the synchronization, and D_r and A_r are the number of detached and active decisions that the requirement r can unify against. On the other hand, when stating that

¹Another reason for the fact that a number of decisions are kept active for each component is that it provides some limited backwards compatibility with the old unification logic that requires all decisions to be active. Thus, planning with detached decision can be seen as an extra service.

at least one decision needs to be active in a synchronization it is only necessary to try

$$\prod_{r=1}^n (D_r + A_r) - \prod_{r=1}^n (D_r)$$

combinations. The complexity of trying every combination using the former method is $O(d^n)$ in the worst case, where d is the number of synchronization opportunities (or more precisely $O((\frac{d}{n})^n)$ if the decisions are evenly spread as unification opportunities for the requirements). The added limitation on the combinations reduces this to roughly $O(d^{n-1})$ though. This is due to the fact that there is an upper limit to the number of active decisions in the decision network so we can consider the entire problem as a fixed number of $O(d^{n-1})$ problems, one for each active decision.

A simple proof of this can be created with the help of the binomial theorem if we assume that each component has an equal number of active and detached decisions, so that $D_r = D$ and $A_r = A$, then:

$$\begin{aligned} \prod_{r=1}^n (D_r + A_r) - \prod_{r=1}^n (D_r) &= (D + A)^n - D^n \\ &= \sum_{r=0}^n \binom{n}{r} D^r A^{n-r} - D^n \\ &= \sum_{r=0}^{n-1} \binom{n}{r} D^r A^{n-r} \end{aligned}$$

Here, the maximum power of D is D^{n-1} , which gives the synchronization the complexity $O(d^{n-1})$.

4.3 Evaluation

In the previous parts of this chapter we have discussed how planning with detached decisions can increase the performance of OMPS from a theoretical point of view. This section continues this discussion by presenting the results of a few simple benchmarks that compares the performance of the system before and after these changes. The performance measurement used is the CPU time required to plan for monitored decisions in the domain, i.e. the rate in which re-planning can occur. Each of the benchmarks were generated by OMPS when planning for decisions in the same way as during activity recognition. It is however impossible to present generic performance measurements that apply to all real world scenarios since the performance depends a lot on the domain and the sensor readings, therefore these benchmarks have been kept simple and only measure the performance for small domains with few synchronizations and components. The benchmarks were selected to provide an empirical evaluation of the performance increase for specific meaningful cases, namely recognition failure, recognition success, and recognition success followed by retraction. Normal usage conditions are typically a combination of such cases. A more realistic, albeit less rigorous benchmark is presented and analyzed in the following chapter.

Each of the benchmarks measure the CPU time required for the OMPS planner to plan for a decision on different reoccurring patterns of sensor readings once per time unit using the five following settings:

Old: Ordinary planning, decisions neither gets frozen nor detached.

F: Decisions are frozen but not detached

F+D(1): Decisions are frozen and detached but one decision is kept active for each component.

F+D(2): Decisions are frozen and detached but two decisions are kept active for each component.

F+D(3): Decisions are frozen and detached but three decisions are kept active for each component.

Finally, each benchmark also contains data about the steadily increasing total number of decisions (active and detached) that have been sensed or monitored.

Benchmark #1

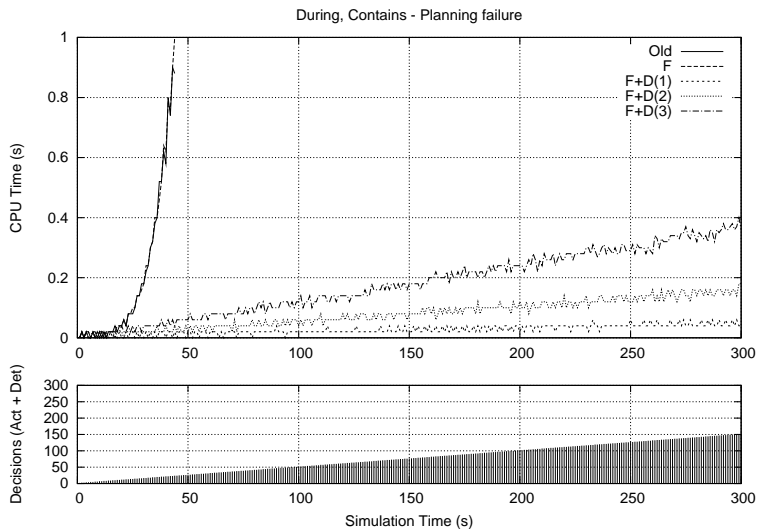


Figure 4.3: Benchmark #1.

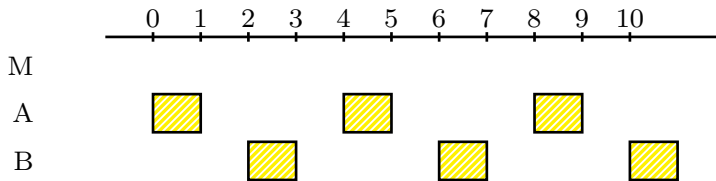


Figure 4.4: Pattern for benchmark #1.

Figure 4.3 and 4.4 shows the results and pattern for benchmark #1. In this benchmark the synchronization that is tried is M DURING A, CONTAINS B. “F+D(1)” gives us the highest performance here while “Old” and “F” performs very poorly, this will be the case in the other benchmarks as well and is due to the constantly growing number of active decisions that results in a high STP propagation cost and a large number of unification combinations that needs to be tried. It is clearly visible how the detached unification logic makes the “F+D” methods scale linearly with the number of decisions and not $O(D^2)$ as it would if all combinations had to be tried.

Benchmark #2

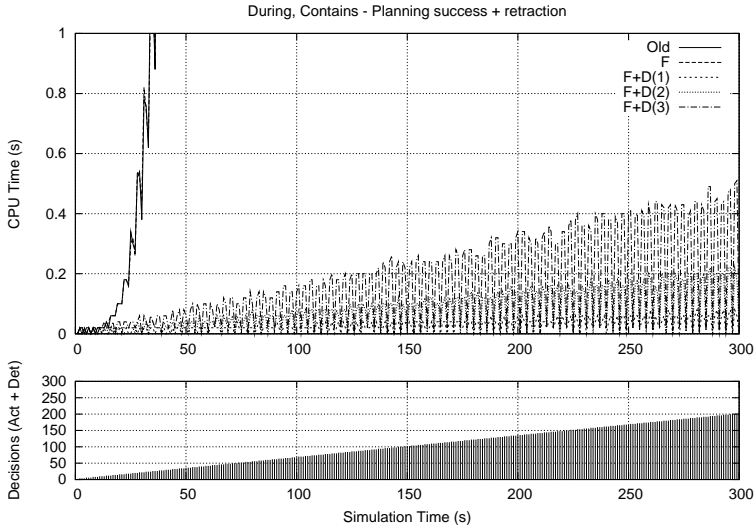


Figure 4.5: Results of benchmark #2.

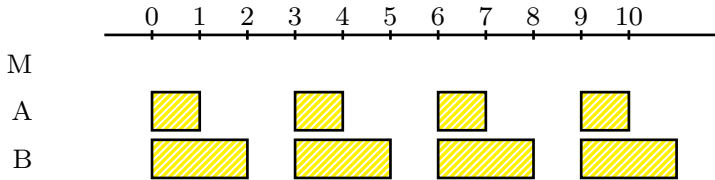


Figure 4.6: Pattern for benchmark #2.

Figure 4.5 and 4.6 shows the results and pattern for benchmark #2. This benchmark uses the same synchronization as in benchmark #1 which it sometimes succeeds to plan for, but in these cases the relative consistency check detects the error and removes the result of the planning. The peaks of the “F+D” curves are the times in which the planning fails and all combinations have to be tried, the valleys on the other hand occur when planning succeeds and is back-tracked. The total number of decisions grow quicker in this benchmark than it does in benchmark #1 since the period of the pattern in Figure 4.6 is shorter than the one in Figure 4.4.

Benchmark #3

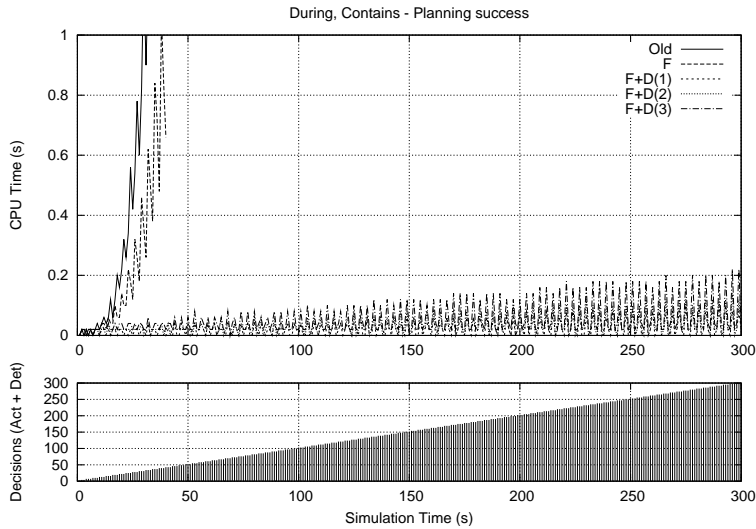


Figure 4.7: Results of benchmark #3.

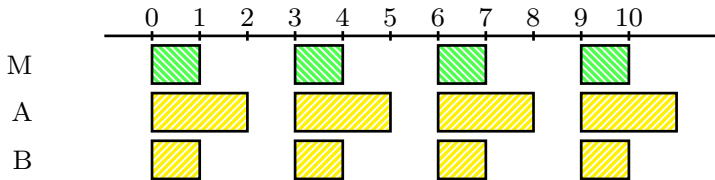


Figure 4.8: Pattern for benchmark #3.

Figure 4.7 and 4.8 shows the results and pattern for benchmark #2. In this benchmark the planning succeeds at some occasions due to the fact that “M” can be synchronized so that it occurs DURING “A” and CONTAINS “B”. This is also the cause for the fact that the decision count grows so rapidly in this benchmark when comparing to the previous two benchmarks. It can be seen that “F” slightly outperforms “Old” in this benchmark, this is due to the fact that the successful planning adds decisions used only for unification which then gets removed during the freezing in “F” but not when using the “Old” method. As such, “F”’s decision network is slightly smaller than the one in “Old”. Furthermore, it can be seen that all “F+D” methods performs well, this is due to the fact that each monitored decision is required to occur after another monitored decision on the same component. Therefore, OMPS can assess that there is only one instance of “A” that is applicable for the synchronization.

Benchmark #4

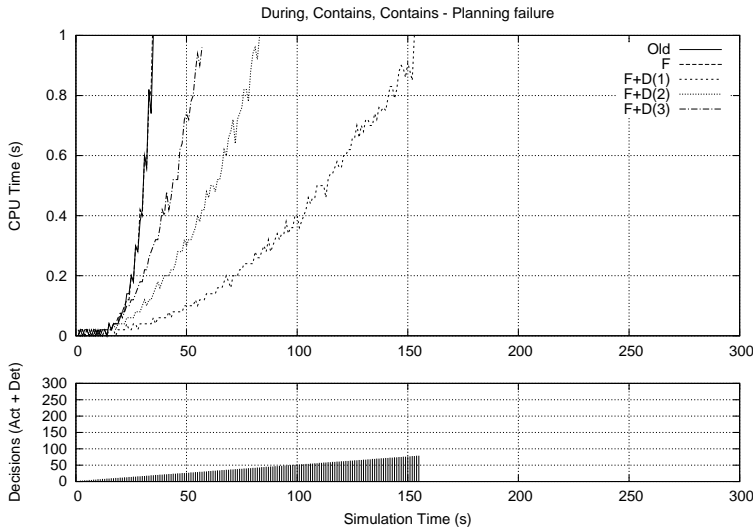


Figure 4.9: Results of benchmark #4.

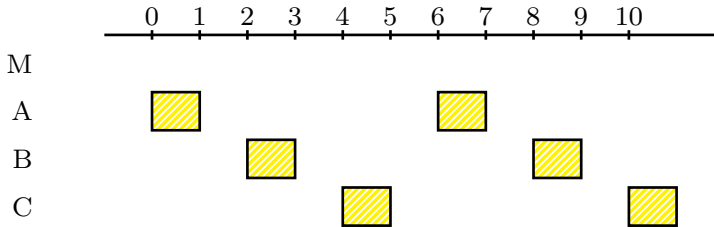


Figure 4.10: Pattern for benchmark #4.

Figure 4.9 and 4.10 shows the results and pattern for benchmark #4. In this benchmark the monitored decision “M” synchronizes against three decisions, “A”, “B” and “C” so that “M” occurs DURING “A” and CONTAINS “B” and “C”. This synchronization however fails at all times due to the structure of the sensed pattern. As such, this benchmark corresponds to #1 with the exception in the fact that it has three requirements. This is also why the “F-D” methods’ planning times grows quadratically with respect to the number of decisions sensed. This can be understood by recalling that the complexity of synchronizing a decision is in the worst case a $O(D^{n-1})$ operation, where n is the number of requirements of the synchronization, and in this case $n = 3$.

Benchmark #5

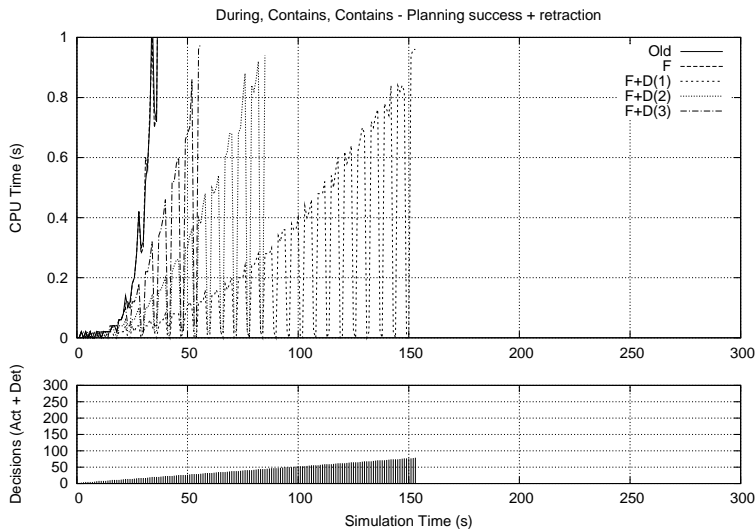


Figure 4.11: Results of benchmark #5.

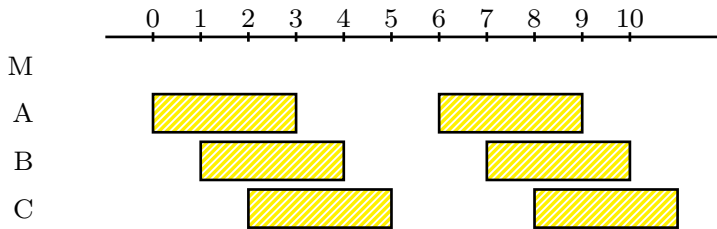


Figure 4.12: Pattern for benchmark #5.

Figure 4.11 and 4.12 shows the results and pattern for benchmark #5. This benchmark corresponds to benchmark #2 with the exception that it uses the synchronization specified in benchmark #4. As such, it continuously tries to plan for the decision “M” which sometimes succeeds, but in those cases the decision is removed again since it fails the relative consistency check.

Benchmark #6

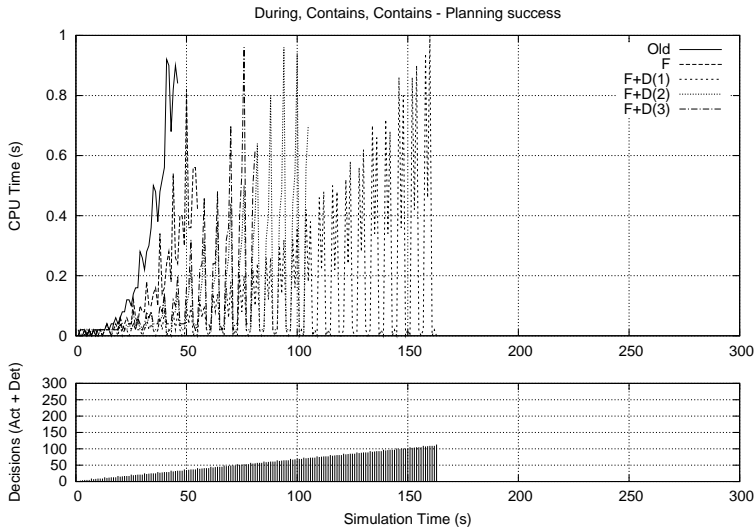


Figure 4.13: Results of benchmark #6.

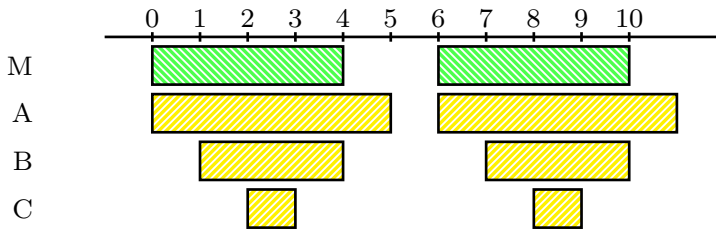


Figure 4.14: Pattern for benchmark #6.

Figure 4.3 and 4.3 shows the results and pattern for benchmark #6. This benchmark corresponds to benchmark #2 with the exception that it uses the synchronization specified in benchmark #4. Therefore it succeeds to plan for the decision “M” in some cases.

Benchmark #7

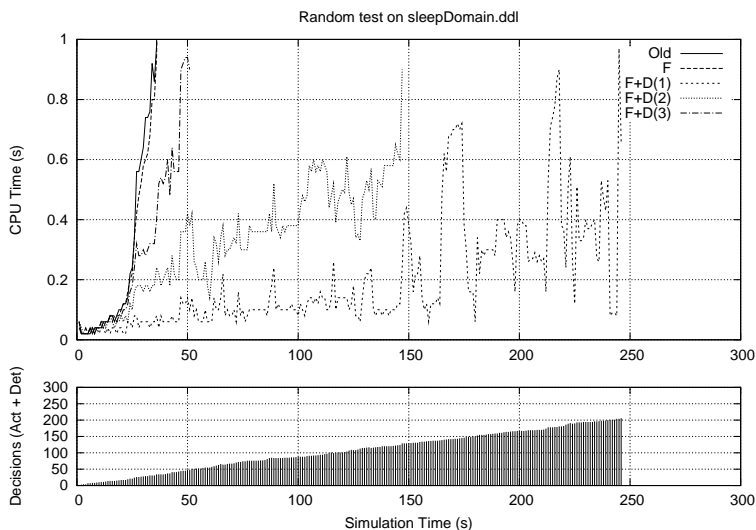


Figure 4.15: Results of benchmark #7.

Figure 4.3 shows the results for benchmark #7. This benchmark is done on random sensor input in a domain with a few different synchronizations similar to the one defined in Appendix A. It is included to show how the performance of the planning process can fluctuate over time as decisions are sensed and planned for using different synchronizations as in a real world scenario. This benchmark also shows how the “F+D” methods perform consistently better than “Old” and “F” even in non-regular domains.

4.4 Summary

This chapter has described how decisions are detached and how detached decisions can be used as targets of synchronizations by the planning process. Planning with detached decisions was proven both theretically and with a series of benchmarks to be far more efficient due to the lowered STP propagation cost and the reduced number of unification combinations that needed to be tried. The benchmarks used did not illustrate the expected performance of the OMPS system for a real world scenario. This will be done in the next chapter which also provides a short overview of the additions made to the OMPS framework during the work on this thesis.

Chapter 5

Implementation and experiments

This chapter contains an overview of the additions made to the OMPS system during the work on this thesis and how these enabled OMPS to perform activity recognition in a real-world scenario that lasted one week.

5.1 Developing a testbed

To be able to perform the work presented in this thesis it was first necessary to do some structural changes in the old activity recognition code written for PEIS. These changes eventually became quite significant and resulted in something that is best considered as an entirely new activity recognition framework. The structure of this framework is illustrated in Figure 5.1.

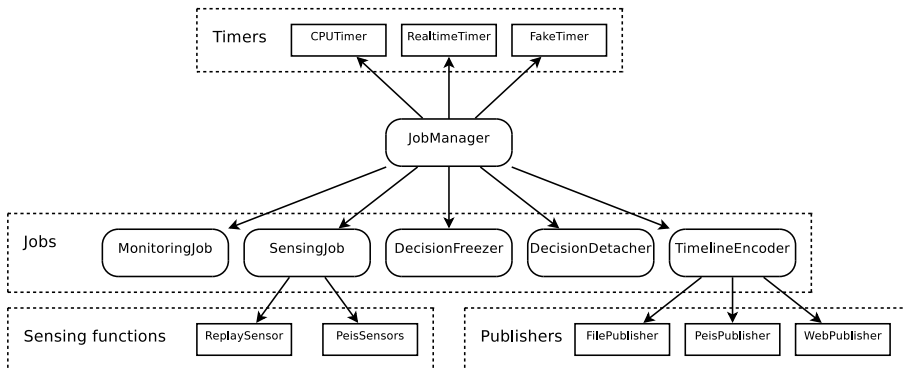


Figure 5.1: The structure of the improved activity recognition framework.

A central concept in the new framework is the abstract concept of a *Job* which represents an activity that is being run at regular intervals and requires

exclusive access to the decision network. Two notable examples of jobs are the *SensingJob* and the *MonitoringJob*. The *SensingJob*'s task is to keep the decision network synchronized with the readings of physical sensors and the *MonitoringJob*'s task is to plan for decisions that should be recognized. Both jobs accomplish this by using the methodologies described in the background chapter. These two jobs are instantiated at a per-component basis, therefore each component in the domain is typically associated with either of these when performing activity recognition.

The following list describes each of the five jobs that were created:

SensingJob Instantiated at a per-component basis for components that represents sensors. The job's responsibility is to read values from a physical sensor and add corresponding decisions to the timeline of the component which the job is associated with.

MonitoringJob Instantiated at a per-component basis for components that represents views of recognized activities. The job's responsibility is to plan for decisions that should be recognized.

TimelineEncoder Responsible for the extraction and publication of timelines for a set of components.

DecisionFreezer Responsible for the freezing of decisions on a set of components in the domain.

DecisionDetacher Responsible for detaching decisions on a set of components in the domain.

Each job is run at a user-defined frequency, and this is done by a *JobManager* whose task is to manage a set of jobs and schedule their access to the decision network. The time instants for which a job is being run is determined by the job's frequency, and each time a job is being run its next start time is determined. For those cases in which two or more jobs want to run at the same time the job that has waited the longest time will run first. Therefore, the scheduling will never lead to starvation, although it can of course be the case that a job cannot be run at its desired time due to high load.

The scheduling process for the jobs makes use of a *timer*, in total there are three types of timers developed to cover the needs of real world activity monitoring and its simulation. The first type of timer is called a *RealtimeTimer* and is used to monitor activities in real world scenarios using physical sensors. For simulation purposes on the other hand, there are two timers to choose from, namely a *CPUTimer* and a *FakeTimer*. The *CPUTimer* allows simulations to be run faster than realtime but misses deadlines in all of those cases in which the *RealtimeTimer* would due to the CPU load. The *FakeTimer* on the other hand also runs faster than realtime but always meets all deadlines. The first two mentioned timers can also be scaled so that their time runs faster or slower with

respect to the real world time (or to put it differently, how many milliseconds of real world time a time unit in the simulation corresponds to).

As previously hinted upon it is also possible to simulate activity recognition scenarios, this is accomplished through the use of *sensing functions* that define how sensor readings are retrieved by the SensingJobs. There are two types of sensing functions to choose from, the first of these are the *PEIS-Sensors* which retrieves sensor readings from sensors connected to the PEIS ecology middleware and is used for real world activity recognition. The second type of sensing function does not read values from a physical sensor but instead from a file containing sensor values for given time instants and is called a *replay sensor*. This is also the sensing function that is used during simulation in conjunction either with the CPU Timer or the FakeTimer. There is however no practical difference between using the PEIS-Sensors or the ReplaySensor since the ReplaySensor behaves in the same way and it is still possible to miss sensor readings if the frequency of the SensingJob is too slow etc.

The final component that will be mentioned in this chapter is the *publisher*. Publishers are used by the TimelineEncoder job to relay the result of the activity recognition through a textual or graphical representation of the timelines for a set of components. Three types of publishers were written, namely a *FilePublisher* that writes graphical images of timelines to file (as in Figure 5.2), a *PeisPublisher* that publishes a string representation and a graphical representation of the timelines in PEIS ecology, and finally a *WebPublisher* that acts as a small webserver which one can use to retrieve a graphical view of the timelines.

5.2 Real world scenario

We conclude this chapter with a larger test that intends to show that the contributions presented in this thesis allows the OMPS planner to recognize domestic activities over longer periods of time. The scenario is “real world” in the sense that it consists of recognizing activities performed by a human in a small apartment through the use of a few sensors that could easily be integrated into any modern home. The sensory input is however simulated just as for the performance benchmarks in section 4.3 due to the fact that during the time of writing, there were no such apartment that could be used exclusively for this thesis during such a long time span. The activity recognition is performed over a simulated time span of one week. This test could have been performed in a real environment with real sensors though since it was constructed for the “PEIS Home” at Örebro University which is equipped with the necessary hardware simulated in this scenario.

The activities that are recognized in this scenario are high-level activities such as “Eating” or “Watching-TV”, and require sensed decisions like the “NightLight” being “On”. All of these activities have one property in common which is the fact that they do not change very fast. Therefore one time unit in the recognition process is set to correspond to one minute of real world time,

this results in $7 * 24 * 60 = 10080$ time units for the entire week as in the figure. Likewise, the sensors are updated and the decisions are planned for once each minute as well.

In short the scenario consists of recognizing the activities of a person that lives in an apartment and follows an approximately equal routine each day, which consists of waking up, eating, going to work, eating again, watching TV, and sleeping. This behavior has been used as a base for modeling the inputs of different sensors, e.g. when the person comes home from work he arrives at the “Entrance” of his home and then walks into the “Living room” where he watches TV for a while before continuing into the “Kitchen” to cook some food. Thus, the readings have not been created exclusively for the purpose of successfully monitoring them since they follow a real-world logic as well (e.g. the human always passes through the living room when going from the kitchen to the bedroom since the rooms are connected in that way in the home). The same pattern of sensor readings are repeated each day but with a small amount of randomness to the time of their occurrence, e.g. it is possible that the human watches the TV between 19:00–20:00 one day and then 19:20–19:50 the next day. It is believed that the problem of recognizing activities with the goal of assisting in the diagnosis of sleeping disorders is similar to the problem solved here. Similar with respect to the longer time frame of the reasoning and the number of events that take place.

The scenario intends to show that the changes made to OMPS made it able to recognize real world activities over longer periods of time with respect to the number of decisions being sensed and the number of decisions being monitored (with synchronizations usable in the real world). As previously explained, there is little practical difference between simulating events and performing a physical test when it comes to evaluating the performance of the reasoning. The intention with the scenario is not to show that the activity recognition is stable or which and when activities can be recognized with respect to (sometimes erratic) real world human behavior.

The five following sensors are used as inputs for the activity recognition process in this scenario:

Bed Senses when the bed is occupied.

KTRfid RFID reader mounted in the kitchen table, senses when a dish is on the table.

Location Keeps track of the humans position within the apartment.

NightLight Senses if the nightlight besides the bed is on or off.

Stove Represents a stove in the kitchen of the apartment and senses whatever it is on or off.

TimeOfDay Keeps track of the different times of the day, i.e. morning, afternoon, evening and night.

During the day a number of activities are monitored with the help of these sensors, for instance when the “Human” is “Cooking”, “Eating” or “WatchingTV”. There is also a higher level view of the human’s activities called “HumanAbstract” that assesses which type of meal the human is having, e.g. “Breakfast” and when the human is “Sleeping” or taking a “Nap”. All synchronizations that were specified for this domain can be seen in Appendix A which shows the domain definition file for the scenario. The timelines from this test are included as Figure 5.2, although, due to the scenario’s long time span, it is not possible to tell from the figure which decisions were sensed or planned for.

The performance of the OMPS planner during this test is shown in Figure 5.3. In this figure the long-term performance seems to be more or less unaffected by the addition of new decisions. This performance cannot entirely be attributed to the detaching logic presented here though since an additional check was added during the writing of this thesis. The added check filters out some combinations of targets that are obviously not going to satisfy a synchronization (e.g. incorrect start ordering in a DURING-CONTAINS synchronization). In short, the check works by comparing the maximum and minimum admissible distances between time instants in expanded decisions to the equivalent distances retrieved from the synchronization targets of each combination. It should be mentioned however that this check would not have been able to provide long term monitoring on its own without the detaching logic. Another reason for the fact that the performance is good in this case is because the domain is more mixed and all decisions are not applicable as targets of all synchronizations. Naturally, the performance will decrease eventually with the number of decisions that are added during the planning, but it is believed that this test shows that the changes made to OMPS made it capable of monitoring activities during longer periods of time for reasonably sized scenarios.

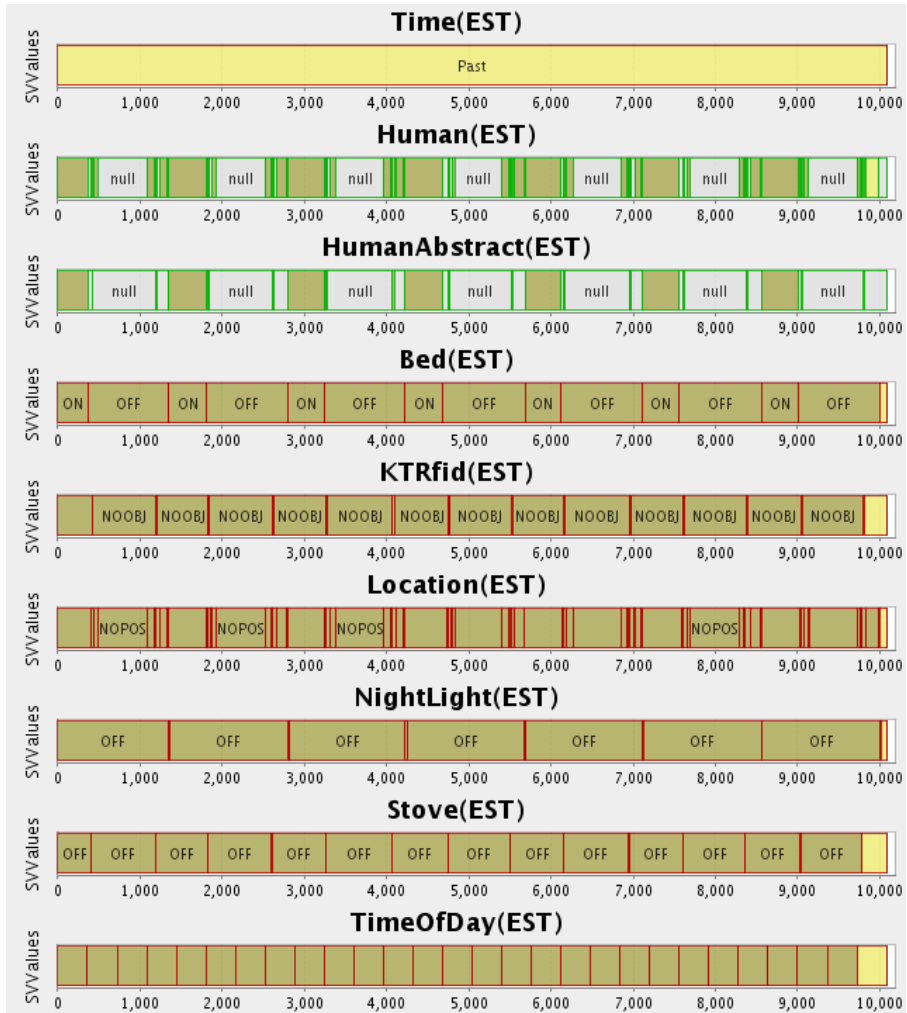


Figure 5.2: Resulting timelines for a set of components containing the sensed and monitored decisions for a real world scenario that has a duration of one week, the time unit used is minutes.

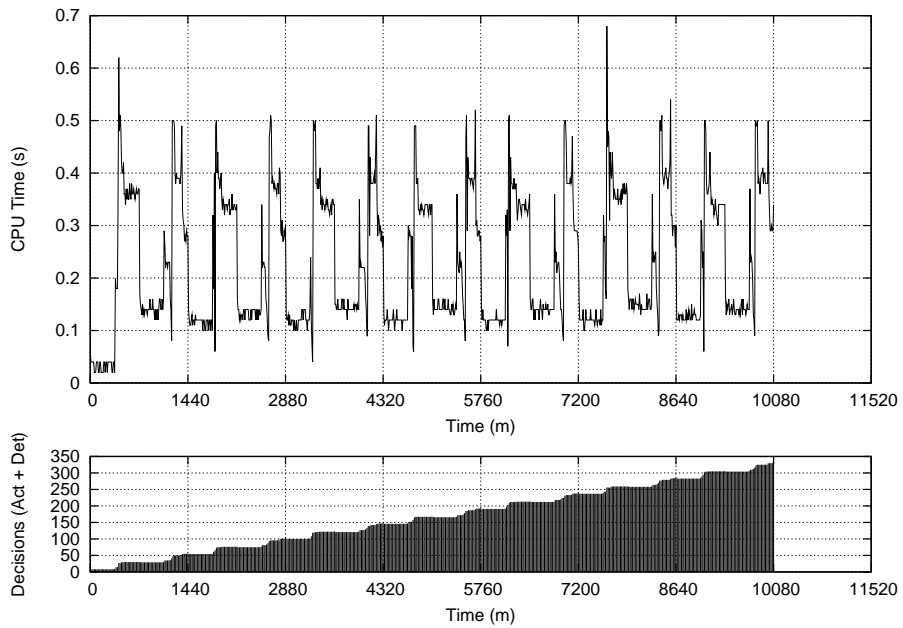


Figure 5.3: Performance during the real world scenario.

Chapter 6

Conclusion

6.1 Summary

This thesis has attempted to solve two particular problems previously encountered when using the OMPS planner for domestic activity recognition, namely the problem of ensuring consistency over time when dealing with sensor readings with uncertain future temporal evolutions and the problem of keeping the reasoning tractable when recognizing activities over longer periods of time.

The solution to the problem of ensuring consistency over time was to implement two algorithms as described in Chapter 3. The first of these algorithms was named the “Absolute consistency check” and assessed that the successful recognition of an activity through the successful planning of its decision did not directly limit the temporal evolution of any other decision previously present in the decision network. If so, the recognition of the activity was to be delayed until its decision could be safely added to the decision network. The second check was similar to the first one and was called the “Relative consistency check”. Unlike the first one, this check instead made sure that the planning of a decision did not put any relative limits on its required decisions. Used together, these two checks ensured that the decision network was kept consistent during the course of time.

Due to the restrictive nature of these checks it was necessary to “Freeze” decisions which consisted of removing their temporal flexibility in order for the planning process to be able to plan for decisions that would otherwise have been considered hazardous. The non-flexible nature of the frozen decisions was then exploited in the solution to the second problem which consisted of ensuring tractability over time. More specifically, the frozen decisions were removed from the decision network through a “detaching” process. This greatly improved the performance of the reasoning due to the fact that the decision network was kept at relatively constant size over time. This could also potentially reduce the reasoning power of the OMPS planner since the detached decisions could not be used as targets of synchronizations anymore. To account for this a

method that was able to use these decisions in the reasoning was implemented which once again enabled the OMPS planner to use detached decision as targets of synchronizations in some circumstances, but this time at a lower cost in terms of performance due to the fact that the detached planning logic was able to reduce the number of unification combinations that needed to be tried significantly.

During the work in this thesis it was also necessary to do some structural changes to the old activity recognition code for the OMPS system as explained in Chapter 5. This chapter also contained a real world test that showed how the changes done to the OMPS system has made it possible to use it for activity recognition during a period as long as one week while maintaining reasonable performance.

To summarize, the important contributions of the work done during this thesis are the following:

- Two problems that are likely to arise when using a temporal planning system to recognize activities in a domestic environment were described, namely maintaining future consistency and performance. A solution to these problems was proposed and implemented.
- Future consistency is assessed by performing flexibility analysis on the simple temporal network.
- Performance is increased by introducing a pruning method that reduces the computational cost of recognizing activities with temporal planning systems.
- Existing ad-hoc activity recognition code in OMPS's code base was refactored in order to make it more modular and extensible. Some additional functionality was also implemented, such as support for simulating activity recognition in a realistic manner.

The conclusion is that the contributions presented in this thesis have solved the two problems that they intended to solve and have furthered the previous work done on using OMPS for activity recognition. There are still some remaining work that can be done though, as will be described in the next section.

6.2 Future Work

There are a number of additions that could be done to the OMPS system to further increase its performance during long-term monitoring. As described previously the performance of the system was mainly governed by the STP propagation cost and the number of combinations that needs to be tried during unification. The current implementation uses Floyd-Warshall's algorithm to solve the STP, which is a proven algorithm but not the most efficient when there

are relatively few constraints in the temporal problem [18]. It could therefore prove worthwhile to create an OMPS implementation of some more efficient STP solver like the Δ STP-Solver [18] or P³C [12]. Work could also be spent on reducing the number of combinations that needs to be tried by implementing some more advanced form of filtering on them, for instance by marking subsets of requirements in synchronizations as impossible when this is the case (e.g. it is possible that a high-order synchronization can never be satisfied with a particular set of target decisions for a subset of its requirements). The additional check that was added during the writing of this thesis (as described section 5.2) could also be tested and described in a more formal way, even though it has been proven in practical tests.

When it comes to ensuring consistency over time it is hard to suggest any concrete improvements to the work done in this thesis. Although changes can certainly be made, for example, the constraint checking could be made less restrictive by adapting the relative and the absolute consistency checks to only account for the temporal evolution of the sensed decisions. This would however break the generic nature of the checks, but might still be desirable from a practical point of view.

Finally, it can also be worthwhile to explore how different data driven approaches can be used in conjunction with OMPS's knowledge driven one. For example, the requirements of the synchronizations in the domain could be trained from recorded sensory data with some statistical method. Such a work could also include an evaluation of how suitable the system is for different individuals, in this thesis the recognized activities have been generic ones giving a high level view of human behavior. Data driven methods can be expected to capture details that makes the system more user dependent.

Bibliography

- [1] J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.
- [2] Shailendra K. Bhonsle, Amarnath Gupta, Simone Santini, Marcel Worring, and Ramesh Jain. Complex visual activity recognition using a temporally ordered database. In *VISUAL '99: Proceedings of the Third International Conference on Visual Information and Information Systems*, pages 719–726, London, UK, 1999. Springer-Verlag.
- [3] A. Cesta, G. Cortellessa, M.V. Giuliani, F. Pecora, M. Scopelliti, and L. Tiberio. 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*, 2007.
- [4] A. Cesta and S. Fratini. The Timeline Representation Framework as a Planning and Scheduling Software Development Environment. In *Proc. of 27th Workshop of the UK Planning and Scheduling SIG*, 2008.
- [5] Tanzeem Choudhury, Gaetano Borriello, Sunny Consolvo, Dirk Haehnel, Beverly Harrison, Bruce Hemingway, Jeffrey Hightower, Predrag “. Klasnja, Karl Koscher, Anthony Lamarca, James A. Landay, Louis Legrand, Jonathan Lester, Ali Rahimi, Adam Rea, and Danny Wyatt. The mobile sensing platform: An embedded activity recognition system. *IEEE Pervasive Computing*, 7(2):32–41, 2008.
- [6] M. Csikszentmihalyi and R. Larson. Validity and reliability of the experience-sampling method. *J Nerv Ment Dis*, 175(9):526–536, September 1987.
- [7] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artif. Intell.*, 49(1-3):61–95, 1991.
- [8] S. Fratini, F. Pecora, and A. Cesta. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences*, 18(2):231–271, 2008.

- [9] Norbert Györbíró, Ákos Fábrián, and Gergely Hományi. An activity recognition system for mobile phones. *Mob. Netw. Appl.*, 14(1):82–91, 2009.
- [10] V. Jakkula, DJ Cook, and AS Crandall. Temporal pattern discovery for anomaly detection in a smart home. In *Proc. of the 3rd IET Conf. on Intelligent Environments(IE)*, pages 339–345, 2007.
- [11] D.J. Patterson, D. Fox, H. Kautz, and M. Philipose. Fine-grained activity recognition by aggregating abstract object usage. In *Proc. of the 9th IEEE International Symposium on Wearable Computers*, 2005.
- [12] L’eon Planken, Mathijs de Weerd, and Roman van der Krogt. P3c: A new algorithm for the simple temporal problem. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-08)*, pages 256–263, 2008.
- [13] M.E. Pollack, L. Brown, D. Colbry, C.E. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan, and I. Tsamardinos. Autominder: an intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems*, 44(3-4):273–282, 2003.
- [14] A. Saffiotti, M. Broxvall, M. Gritti, K. LeBlanc, R. Lundh, Rashid J., B.S. Seo, and Y.J. Cho. The PEIS-ecology project: vision and results. In *Proc of the IEEE/RSJ Int Conf on Intelligent Robots and Systems (IROS)*, Nice, France, 2008.
- [15] Dairazalia Sanchez, Monica Tentori, and Jesus Favela. Hidden markov models for activity recognition in ambient intelligence environments. In *ENC ’07: Proceedings of the Eighth Mexican International Conference on Current Trends in Computer Science*, pages 33–40, Washington, DC, USA, 2007. IEEE Computer Society.
- [16] Emmanuel M. Tapia, Stephen S. Intille, and Kent Larson. Activity recognition in the home using simple and ubiquitous sensors. *Pervasive Computing*, pages 158–175, 2004.
- [17] Christian Wojek, Kai Nickel, and Rainer Stiefelhagen. Activity recognition and room-level tracking in an office environment. In *Multisensor Fusion and Integration for Intelligent Systems, 2006 IEEE International Conference on*, pages 25–30, 2006.
- [18] Lin Xu and Berthe Y. Choueiry. A new efficient algorithm for solving the simple temporal problem. *Temporal Representation and Reasoning, International Symposium on*, 0:212, 2003.
- [19] Chun Zhu and Weihua Sheng. Multi-sensor fusion for human daily activity recognition in robot-assisted living. In *HRI ’09: Proceedings of the 4th*

ACM/IEEE international conference on Human robot interaction, pages 303–304, New York, NY, USA, 2009. ACM.

Appendix A

WeekTest.ddl

```
1 DOMAIN WeekTest {
2
3   %-----
4   % Type definitions
5   %-----
6   COMP_TYPE PEIS.MonitoringNew.MonitoringJob SV_Human
7   (Eating(), Cooking(), WatchingTV(), Out(), InBed(), BedReading(), None()) {
8     VALUE Eating() [1,+INF]
9     MEETS {None()}
10    VALUE Out() [1,+INF]
11    MEETS {None()}
12    VALUE Cooking() [1,+INF]
13    MEETS {None()}
14    VALUE WatchingTV() [1,+INF]
15    MEETS {None()}
16    VALUE InBed() [1,+INF]
17    MEETS {None()}
18    VALUE BedReading() [1,+INF]
19    MEETS {None()}
20    %% "Null" action
21    VALUE None() [1,+INF]
22    MEETS {Eating(),Cooking(),WatchingTV(),Out(),InBed(),BedReading(),None()}
23  };
24
25  COMP_TYPE PEIS.MonitoringNew.MonitoringJob SV_HumanAbstract
26  (Breakfast(), None(), Sleeping(), Lunch(), Dinner(), Nap()) {
27    VALUE Breakfast() [1,+INF]
28    MEETS {None()}
29    VALUE Nap() [1,+INF]
30    MEETS {None()}
31    VALUE Lunch() [1,+INF]
32    MEETS {None()}
33    VALUE Dinner() [1,+INF]
34    MEETS {None()}
35    VALUE Sleeping() [1,+INF]
36    MEETS {None()}
37    VALUE None() [1,+INF]
38    MEETS {Breakfast(), Sleeping(), Lunch(), Dinner(), Nap(), None()}
39  };
40
41  COMP_TYPE PEIS.MonitoringNew.SensingJob SE_Stove
42  (ON(), OFF(), open(), closed(), error(), working()) {
43    VALUE ON() [1,+INF]
44    MEETS {working()}
45    VALUE OFF() [1,+INF]
```

```

46     MEETS {working()}
47     VALUE open() [1,+INF]
48     MEETS {working()}
49     VALUE closed() [1,+INF]
50     MEETS {working()}
51     VALUE error() [1,+INF]
52     MEETS {working()}
53     VALUE working() [1,+INF]
54     MEETS {open(),closed(),error(),working(),ON(),OFF()}
55 };
56
57 COMP_TYPE PEIS.MonitoringNew.SensingJob SE_Location
58 (COUCH(),ENTRANCE(),KITCHENTABLE(),KITCHEN(),LIVINGROOM(),NOPOS()) {
59     VALUE COUCH() [1,+INF]
60     MEETS {NOPOS()}
61     VALUE ENTRANCE() [1,+INF]
62     MEETS {NOPOS()}
63     VALUE KITCHENTABLE() [1,+INF]
64     MEETS {NOPOS()}
65     VALUE KITCHEN() [1,+INF]
66     MEETS {NOPOS()}
67     VALUE LIVINGROOM() [1,+INF]
68     MEETS {NOPOS()}
69     VALUE NOPOS() [1,+INF]
70     MEETS {COUCH(),ENTRANCE(),KITCHENTABLE(),KITCHEN(),LIVINGROOM(),NOPOS()}
71 };
72
73 COMP_TYPE PEIS.MonitoringNew.SensingJob SE_KTRfid (DISH(),NOOBJ()) {
74     VALUE DISH() [1,+INF]
75     MEETS {NOOBJ()}
76     VALUE NOOBJ() [1,+INF]
77     MEETS {DISH(),NOOBJ()}
78 };
79
80 COMP_TYPE PEIS.MonitoringNew.SensingJob SE_Bed (ON(),OFF()) {
81     VALUE ON() [1,+INF]
82     MEETS {OFF()}
83     VALUE OFF() [1,+INF]
84     MEETS {OFF(),ON()}
85 };
86
87 COMP_TYPE PEIS.MonitoringNew.SensingJob SE_NightLight (ON(),OFF()) {
88     VALUE ON() [1,+INF]
89     MEETS {OFF()}
90     VALUE OFF() [1,+INF]
91     MEETS {OFF(),ON()}
92 };
93
94 COMP_TYPE PEIS.MonitoringNew.SensingJob SE_Time
95 (morning(),afternoon(),evening(),night()) {
96     VALUE morning() [1,+INF]
97     MEETS {night()}
98     VALUE afternoon() [1,+INF]
99     MEETS {night()}
100    VALUE evening() [1,+INF]
101    MEETS {night()}
102    VALUE night() [1,+INF]
103    MEETS {morning(),afternoon(),evening(),night()}
104 };
105
106 %-----
107 % Component definitions
108 %-----
109 COMPONENT Human : SV_Human;
110 COMPONENT HumanAbstract : SV_HumanAbstract;

```

```

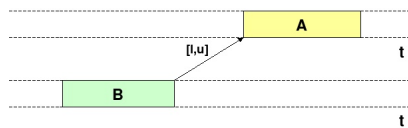
111 COMPONENT Stove : SE_Stove;
112 COMPONENT Location : SE_Location;
113 COMPONENT KTRfid : SE_KTRfid;
114 COMPONENT Bed : SE_Bed;
115 COMPONENT NightLight : SE_NightLight;
116 COMPONENT TimeOfDay : SE_Time;
117
118 %-----
119 % Synchronizations (Domain theory)
120 %-----
121 COMPONENT Human : SV_Human {
122   VALUE Cooking() {
123     EQUALS Stove ON(),
124     DURING [0,+INF][0,+INF] Location KITCHEN()
125   }
126   VALUE Eating() {
127     EQUALS KTRfid DISH(),
128     DURING [0,+INF][0,+INF] Location KITCHENTABLE()
129   }
130   VALUE WatchingTV() {
131     EQUALS Location COUCH()
132   }
133   VALUE Out() {
134     MET-BY Location ENTRANCE(),
135     DURING [0,+INF][0,+INF] Bed OFF(),
136     EQUALS Location NOPOS()
137   }
138   VALUE InBed() {
139     DURING [0,+INF][0,+INF] Location NOPOS(),
140     EQUALS Bed ON()
141   }
142
143 };
144
145 COMPONENT HumanAbstract : SV_HumanAbstract {
146   VALUE Breakfast() {
147     EQUALS Human Eating(),
148     DURING [0,+INF][0,+INF] TimeOfDay morning()
149   }
150   VALUE Lunch() {
151     EQUALS Human Eating(),
152     DURING [0,+INF][0,+INF] TimeOfDay afternoon()
153   }
154   VALUE Dinner() {
155     EQUALS Human Eating(),
156     DURING [0,+INF][0,+INF] TimeOfDay evening()
157   }
158   VALUE Sleeping() {
159     %DURING [0,+INF][0,+INF] Human InBed(),
160     %EQUALS NightLight ON()
161     EQUALS Human InBed()
162   }
163   VALUE Nap() {
164     AFTER [0,+INF] HumanAbstract Lunch(),
165     EQUALS Human WatchingTV()
166   }
167 };
168 }

```

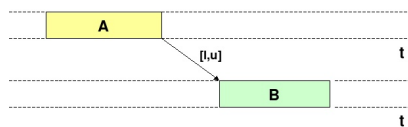

Appendix B

Temporal constraints

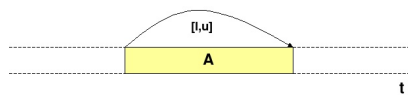
After



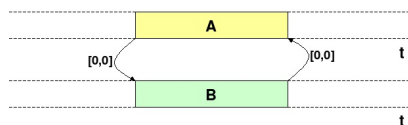
Before

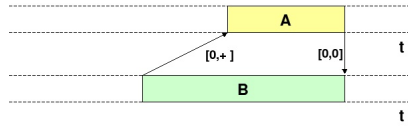
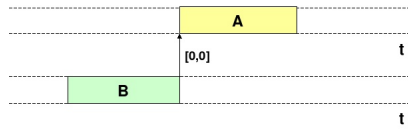
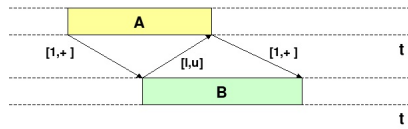
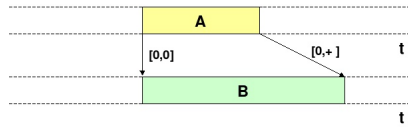
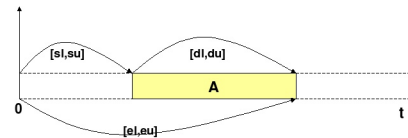
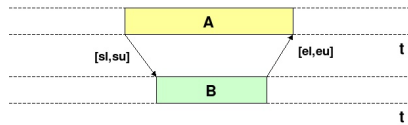
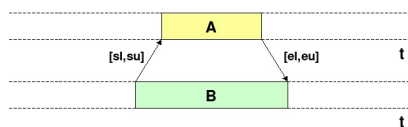


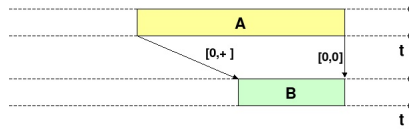
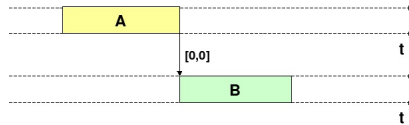
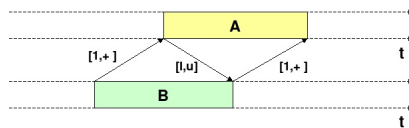
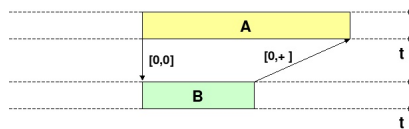
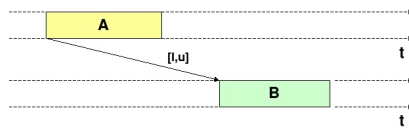
Duration



Equals



Finishes**MetBy****Overlaps****Starts****At****Contains****During**

FinishedBy**Meets****OverlappedBy****StartedBy****StartStart****Ends-During**