Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

An Approach for Activity Recognition in Buildings based on Temporal HTN Planning

Ivaylo Spasov

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Dr. Ilche Georgievski
Commenced:	August 1, 2022
Completed:	February 1, 2023

Abstract

Accurate automatic recognition of activities in various contexts is a topic of great interest in the research area of ubiquitous computing. Data-based and knowledge-driven approaches, or a mixture of both, have been developed in the recent years to achieve activity recognition in smart offices, homes of elderly and cognitively impaired and other kinds of buildings. A major problem in this field is the modelling effort, required to adapt a model to a new scenario with new activities, making the practical deployment of such systems often challenging. We discuss how we can apply a planning technique, that is able to represent the hierarchical and structural relationships between activities, namely Hierarchical Task Network (HTN) planning, to some of the state of the art activity recognition models and even improve the gaps in their expressive power. To prove the capability of HTN planning for activity recognition, we propose a generic and simple HTN planning domain model, achieving comparable results with some of the more complex data-driven approaches, which gives a solid reason for further research.

Contents

1 Introduction **13**

2 Background Information **17**
2.1 Classical (AI) Planning 17
2.2 Hierarchical Task Network planning 19
2.3 State-based HTN planning 21
2.4 Planning language 23
2.5 SIADEX 25

3 State of the Art **29**
3.1 Activity recognition by sequence of objects 29
3.2 Activity recognition using hierarchical plans 31
3.3 Ontology-based activity recognition 33
3.4 Hybrid approaches 35

4 Design of Hierarchical Planning Activity Recognition **37**
4.1 Sensor Data 37
4.2 Activity recognition 39
4.3 Hierarchical representation of an activity 40

5 Realisation of Hierarchical Planning Activity Recognition **43**
5.1 Technology stack 43
5.2 Domain for recognising temporal activities 44
5.3 Tasks for recognising activities 46
5.4 Problem instance of the domain 52

6 Validation & Evaluation of Solution **53**
6.1 Experimental setup 53
6.2 Results 55
6.3 Computational load 59

7 Conclusion and Outlook **61**
7.1 Hierarchical planning for activity recognition 61

Bibliography **63**

List of Figures

2.1 An example graph of plan-based searching space for an HTN planning problem P. 21
2.2 An example graph of state-based searching space for an HTN planning problem P. 22

3.1 Hierarchy for recursive bathroom activities 31
3.2 Hierarchical model of activities of daily living (ADL) Having Breakfast as in [NBW] 32
3.3 Ontology for recognising activities in offices [NRA13] 34

4.1 Overview of the system architecture 37
4.2 Mapping of MQ Telemetry Transport (MQTT) messages to predicates 39
4.3 Hierarchical model of washing hands [BHP+06] 41

6.1 Precision for 'none', 'leave house', 'go to bed' and 'use toilet' 56
6.2 Precision for 'take shower', 'get drink', 'prepare breakfast' and 'prepare dinner' . 56
6.3 Recall for 'none', 'leave house', 'go to bed' and 'use toilet' 57
6.4 Recall for 'take shower', 'get drink', 'prepare breakfast' and 'prepare dinner' . . . 57
6.5 Average values for precision, recall and F-score for the different activities 58
6.6 Comparison of the performance of different approaches for House A [AA18] . . 59

List of Listings

2.1 Domain and problem for pouring water between jugs, adapted from [McD00] to PDDL2.1 25
2.2 An Hierarchical Plan Definition Language (HPDL) task [FP20] 26
4.1 Rough description of the process of recognition of activities 40
5.1 A type hierarchy for activity recognition model 44
5.2 Predicates for the domain 45
5.3 Compound task for updating sensor states 47
5.4 Root compound task for the domain 47
5.5 Methods for handling the sensor information sequence 49
5.6 Presence recognition methods 50
5.7 Activities at meeting room recognition methods 51
5.8 An example SIADEX solution plan for a problem 52

Acronyms

ADL activities of daily living. 7, 13

AI Artificial Intelligence. 14

BECM building energy and comfort management system. 14

HDDL Hierarchical Domain Definition Language. 24

HPAR Hierarchical Planning Activity Recognition. 14

HPDL Hierarchical Plan Definition Language. 9, 24

HTN Hierarchical Task Network. 3

IoT Internet of Things. 38

MQTT MQ Telemetry Transport. 7

PDDL Planning Domain Definition Language. 23

RFID radio-frequency-identification. 29

1 Introduction

The vision of *ubiquitous computing*, blending technology with human activity, or better said using technology as a seamless extension to our day to day life, is explored in [Wei99]. Dating back to the now distant 1999, this vision of integrating devices in our surrounding environment in a way that would make us even forget that they are actually an electronic device, is nowadays more relevant than ever. Smartphones have more functionality than a personal computer from 20 years ago, smartwatches are slowly but steadily catching up with smartphones - devices are getting ever smaller and more functional. Moreover, their importance in our daily life is also rising - the modern person has a panic attack when he forgets his smartphone at home.

But it is not just smartphones or smartwatches. We can enjoy a home cinema experience on our high quality smart TV, and that TV has a built-in camera. Many of us have a personal assistant, a smart speaker, to which we can talk to, we can command it to play a certain song for us, we can ask it how cold it is outside, etc. We have smart lamps that we can control from our mobile device, we can turn off the air conditioning in the living room from the comfort of our own bed. In many of the modern homes *ubiquitous computing* (also known as *pervasive computing*) is taking place, with or without our aware consent.

With this amount of data coming from many different sources, all in the area of our home, an interesting question arises. Can we blend this rich information together to further reason about and consequently assist our life? As in, what if the voice command to turn the music off serves not only the purpose to merely cut the playback of music, but to also infer what else could be happening in the context that we do not want to listen to music any more? For example, a pressure sensor in our bed reports that we are laying in it, the lights in our bedroom are switched off and the clock tells us that it is close to midnight - well, it is quite probable that we are about to go to bed, and now our smart speaker suggests us to play a bit of relaxing soft music before it finally switches off. The speaker by itself is not capable of inferring that we are about to go to sleep, but a system watching over the aforementioned sensors can easily instruct it that this is the case, as it is aware of the bigger current context in our bedroom.

In scientific literature, there is a variety of approaches to recognising a set of activities of daily living (ADL) in different buildings, ranging from smart offices to personal homes. It is by itself a broad topic which is of interest to different domains serving various purposes. [PFP+04] reports that recognising and recording ADL is a major issue in elder care. Monitoring manually such activities is time consuming, prone to errors and invasive. Delegating this task to *pervasive computing*, although also sensitive to errors, surely would ease the healthcare professionals and free them from the mundane tracking of who is doing what and allow them to focus on their more important duties as actually assisting and taking care of people. Moreover, errors in the case of automatic recognition would only be a consequence of the system itself rather than a lack of concentration. We can always understand why a computer did something wrong - it is only because it was instructed to do so. With humans, error tracking is tremendously more complex and sometimes impossible.

Knowing what a person is doing at his home is not only of interest to such higher purposes as assisting elders. More often we want to improve and optimise the quality of life of the regular person, regardless of his age. The goal of a building energy and comfort management system (BECM) is to meet the requirements of the occupants for comfort while reducing energy consumption [NRA13], the benefit of which seems nowadays more relevant than ever. Obviously, knowing which activity an occupant is currently performing is of great interest to such a system. If he is currently watching a presentation in the meeting room, his monitor could be turned off until he returns to his workplace.

Automatically detecting ADLs is surely a futuristic vision of a great value to domains of different nature, but its realisation is a complex challenge on many levels. Computers perform incredibly well in a closed world scenario, where the outcome is dependent solely on the input and the algorithm itself - after all, that is what they were designed for. Adapting and using their computing power in the real world, however, is a scenario that shows their drawbacks. We cannot expect the open world environment to behave in a systematic and predictable manner even in relatively isolated situations like our homes. As soon as we leave the comfort and practicality of the closed world assumption, we have to expect a certain degree of uncertainty [Geo15]. Just because we offered the user to listen to relaxing music before falling asleep a hundred times and a hundred times he happily agreed, we cannot be sure that the next time he will also react positively. What if he is having a really bad day and the last thing he wants now is to listen to music? What if this makes him burst in anger and destroy the smart speaker? And it is not only human emotions that are unpredictable, technical devices are bound to malfunction sooner or later and when a sensor reports incorrect data, the inferred activity is most likely not going to be correct as well, etc.

Occasional corrupt sensor data and consequently falsely recognised activity is not of critical damage to every domain - in the field of energy saving buildings, a minor loss in detail can be sacrificed for the greater cause of cost and ease-of-use [NRA13]. However, recognising ADLs is a challenging task even if the network of devices reports only valid data. Users perform activities in various ways and the recognition system must respect this variety [PFP+04]. Moreover, ADLs can be classified in 20 to 30 sets with thousand of instances. Combining thousands of activities with different ways of performing each single one, it is an overwhelming realm of possibilities for a system to handle. Taking this into account, it is easy to understand why research on this topic lacks interconnection and generality [PFP+04].

The possible gaps in sensor data are usually handled by supervised or unsupervised learning approaches that analyse the input data sequences. This branch of approaches is called data-driven, while the other main branch, that of the knowledge-driven approaches, models the contextual information about activities. In more recent researches, a hybrid version, combining data and knowledge reasoning, has emerged. One of the biggest challenges for a real world deployment is the required modelling effort to adapt the system to a given scenario, since the number of users and the modelling effort may be too high to model manually [AA18]. We present the current different approaches in Chapter 3.

The main purpose of this work is to propose a generic framework for recognising ADLs in buildings using a novel *Artificial Intelligence (AI) planning* technique, HTN planning. The background information on HTN planning is thoroughly described in Chapter 2. We call it Hierarchical Planning Activity Recognition (HPAR) system. As input we use raw sequential sensor data. We do not want to make any further assumptions about the sensors apart from their location and that the data they report is either *binary*, that is an on/off state, or *numeric*, as for a numeric temperature value for a

temperature sensor, electricity consumption for a power sensor, etc. By knowing the location of a sensor and its current reported value we can infer whether it is active or not, and by knowing this about every sensor at a given location at a given timestamp, we can finally conclude the activity that is happening at this location for that timestamp.

In order to define an adequate system, our first challenge is to gather raw sensor data in either a real world scenario or simulating sensor readings and a message broker. Next, we have to implement a transformation of the sensor data into a suitable planning language. In order to do this we have to define the key features of a sensor reading in relation to the planning language like their value, their time point, the sequence of the readings. Our ultimate goal is to recognise activities, so we have to define a hierarchy which uses this information to do this. Parallel to developing this temporal HTN representation, we have to carefully choose a planner that would be capable of finding solutions for such problems efficiently. This will aim to show if temporal planning is practically executable, as there is not a wide choice of planners that fulfil our requirements, and if so, to present its advantages in comparison to already existing solutions. The design choices and implementation details are described in Chapter 4 and Chapter 5.

The rest of the paper is structured as follows - in the following Chapter 2, we provide an extensive background summary on AI planning, and more specifically, HTN planning. We then give an overview of the current state of the art approaches to automatic activity recognition in Chapter 3, followed by the design and implementation of HPAR system in Chapter 4 and Chapter 5. We test and evaluate our system in Chapter 6 and conclude our work in Chapter 7.

2 Background Information

To be able to properly introduce and define our solution's design and implementation, a wide range of preliminary definitions are needed, since the domain of *AI planning* is based on formal logic theory. We first elaborate on the general topic of *planning*. In ubiquitous computing, planning is seen as a suitable technique to achieve goal-oriented behaviour [Geo15]. It is capable of handling dynamic and uncertain environments, reasoning about time and parallelism of actions and distribution of devices. All these qualities, as already discussed, are very lucrative to the domain of recognising ADLs. We are not necessarily interested in a certain goal in the process of inferring activities, however the HTN planning concept shifts this goal-oriented behaviour, as will be later discussed.

2.1 Classical (AI) Planning

Classical planning is based on a *state model*, defined over a state space. It has a single initial state, a non-empty set of goal states, and a set of actions that transform each state into another in a deterministic way [BG00]. The following definitions in Section 2.1, Section 2.2, Section 2.3 are adapted from [Geo15].

Definition 2.1.1 (State model)
A state model M is a tuple $\langle S, s_0, S_g, A, \delta \rangle$, where

- *S is the finite and discrete set of states,*
- *$s_0 \in S$ is the initial state,*
- *$\emptyset \neq S_g \subseteq S$ is the set of goals,*
- *A is the finite set of actions,*
- *$\delta : S \times A \rightarrow S$ is the deterministic transition function.*

Action $a \in A$ is applicable in $s \in S \Leftrightarrow (s, a)$ is in the domain of δ.

When an action a is executed in state s, the resulting state is $s' = \delta(s, a) \equiv s[a] = s'$. We can now see how a sequence of actions $a_1, \ldots, a_n$ transforms the state:

$$s[] = s$$
$$s[a_1, \ldots, a_n] = s[a_1][\ldots, a_n] = \ldots = s'[a_n]$$

Given Definition 2.1.1, we are interested in finding such a sequence of actions $a_1, \ldots, a_n$ that transforms the initial state s_0 to any of the goal states $s_g \in S_g$. We call this sequence a *solution*, or a *plan*.

Definition 2.1.2 (Plan)
Let $M = \langle S, s_0, S_g, A, \delta \rangle$ be a state model.

$$\{a_1, \ldots, a_n\} \subseteq A, \pi = a_1, \ldots, a_n \textit{ is a } \textbf{\textit{plan}} \Leftrightarrow s_0[\pi] \in S_g$$

With Definition 2.1.1 and Definition 2.1.2 we have a complete definition of classical planning, and *AI planning* deals with this task computationally [GNT04]. For the purpose of this work, we are from now on interested solely in AI planning, therefore we use these terms interchangeably.

To make the state space feasible for processing of planning problems increasing in size, further conventions are needed - states can only consist of a set of values for propositional variables, that it a propositional variable can be either *true* or *false* [Geo15]. If a propositional value is true, it is a part of of the current state and else it is excluded from it (*closed world assumption*). The definition below is central to the following ones and known as *STRIPS planning problem* [FN71].

Definition 2.1.3 (STRIPS planning problem)
A STRIPS planning problem P^{STRIPS} is a tuple $\langle F, O, I, G \rangle$, where

- *F is the set of propositional variables, also knows as **facts**, **atoms** or **fluents**,*
- *O is the set of operators, where an operator $o \in O = \langle pre(o), add(o), del(o) \rangle$ and $\langle pre(o), add(o), del(o) \rangle \subseteq F$,*
- *$I \subseteq F$ is the initial state,*
- *$G \subseteq F$ is the goal state.*

Definition 2.1.3 holds an implicit Definition 2.1.1. A state $s \in S$ is constituted of such variables $v \in F$, that hold true. Respectively, $\bar{v} \in F \setminus s$ have value false. I corresponds to the initial state s_i and G to the set of goals S_g. Now we can see how the set of actions A in the state model is further refined by the STRIPS planning problem. An action $a \in A$ is equivalent to an operator $o \in O$, that is $\langle pre(o), add(o), del(o) \rangle$. Each of this components is a subset of the propositional variables. $pre(o)$ is the precondition of o, that is the variables that need to hold true in the current state, so that we can apply o. This would result in the successor state by adding $add(o)$ and subtracting $del(o)$:

$$\text{We are in state } s \text{ and } pre(o) \subseteq s \Rightarrow \text{we can apply } s[o] = (s \cup add(o)) \setminus del(o) = s'$$

A series of actions $a_1, \ldots, a_n$ is a plan π, if $pre(a_{i+1}) \subseteq s_i$, as s_i is the resulting state from $s_{i-1}[a_i]$ for $i = 1, \ldots, n$. In particular, $s_0 = I$ is the initial state and $s[\pi] \in S_g = G$, that is we reach a goal state by executing the series of actions, beginning in the initial state.

2.2 Hierarchical Task Network planning

Now we have all the basic tools to define the planning technique used in our work, namely HTN planning. We mentioned that when designing a system to recognise ADLs, we are not interested in a particular goal, as this is the main objective of classical planning. When we deploy our system, there is no particular activity that we aim to recognise, and the process is not over once we have identified a given activity - we want to keep looking for the next one as long as we still have data to process.

The domain knowledge in HTN planning consists of task networks of *primitive* and *compound* tasks. The initial state equates to an initial task network and the optional goal is a task network to be achieved. It is in these task networks that we can encode hierarchical relationships. They represent a hierarchy of tasks, and a task itself can be executed if the task is primitive, or decomposed in subtasks, again compound or primitive, thus representing a hierarchy. A solution contains only primitive tasks which can be applied to the initial world state, that is a decomposition of the initial task network to a primitive one.

Definition 2.2.1 (HTN planning language)
The HTN planning language is a first-order language containing the sets C, V, P, Q, where

- *C is a finite set of constant symbols,*
- *V is an infinite set of variable symbols,*
- *P is a finite set predicate symbols, where each $p \in P : p := (t_1, \ldots, t_k)$ is a list of **terms** and each term $t_i \in C \cup V$, that is a **constant** or **variable** symbol. A predicate can be true or false, and a predicate is **ground**, if it does not contain variable symbols,*
- *Q is the set of all predicates.*

With respect to Definition 2.1.1, a state $s \in S$ is now a set of ground predicates $\subseteq 2^Q$.

As primitive tasks represent the actions that can be executed directly, if their preconditions are met, they can be mapped to the definition of an *operator* from Definition 2.1.3 as follows:

Definition 2.2.2 (Operator)
Let $s \subseteq 2^Q$ be a state as in Definition 2.2.1. An operator o is a triple $\langle p(o), pre(o), eff(o) \rangle$, where

- *$p(o)$ is a primitive task,*
- *$pre^+(o)$ and $pre^-(o)$ are the positive and negative preconditions of o, that is o is applicable in s when $pre^+(o) \subseteq s$ and $pre^-(o) \cap s = \emptyset$*
- *$eff^+(o)$ and $eff^-(o)$ are the positive and negative effects of o, that is $s[o] = (s \cup eff^+(o)) \setminus eff^-(o) = s'$*

Hierarchical planning breaks with the tradition of classical planning [GNT04], and one of the main reasons for that is the possibility to encode domain-specific knowledge through compound tasks. This is a point that will be later further elaborated, as the extension is controversial amongst the community of AI planning [GA15], and our implementation takes advantage of this expressive power. By implementing rich domain-specific knowledge, we provide more guidance for the solving

of a problem than was envisioned for classical planning. Going back to the compound task, it is an expression $t_c(t_1, \ldots, t_k)$, where $t_c \in T_c$ and T_c is a finite set of compound symbols, t_i are terms. We denote the union of the sets of primitive and compound task names as T_n. The following definitions complete the setup for an HTN planning problem.

Definition 2.2.3 (Task network)
A task network is pair $\langle T, \Psi \rangle$, where T is a finite set of (compound or primitive) tasks, and Ψ is a set of constraints.

A method maps, or decomposes, a compound task to a refined task network.

Definition 2.2.4 (Method)
A method m is a pair $\langle c(m), pre(m), tn(m) \rangle$, where $c(m)$ is a compound task (to be decomposed), $pre(m) \in 2^Q$ is a precondition, and tn(m) is a task network. The subsets $pre^+(m)$ and $pre^-(m)$ are the positive and negative preconditions of m, similar to Definition 2.2.2.

Definition 2.2.5 (HTN planning problem)
An HTN planning problem is a tuple $\langle Q, O, M, tn_0, s_0 \rangle$, where

- *Q is a finite set of predicates,*
- *O is a finite set of operators,*
- *M is a finite set of methods,*
- *tn_0 is an initial task network,*
- *s_0 is the initial state.*

As evident from Definition 2.2.5, a problem in HTN planning does not necessarily include a goal task network to be reached. This is another main difference from classical planning - it is about searching for a sequence of actions that decomposes the initial task network into a primitive one beginning from the initial state [Geo15]. Given an HTN planning problem P, a plan is a solution if it includes an operator sequence (a sequence of primitive tasks) applicable in s_0 by decomposing tn_0. This is very suitable to the environment setup we are facing, as we want to recognise activities beginning from the raw sensor data, representing the initial state. Then by developing appropriate task operators, methods and task networks, we want to achieve a decomposed primitive task network, equivalent to the recognised activities.

Now that we have a broader picture of HTN planning, it is time to look at how we can solve problems in this setup and extend our definitions accordingly. Problems in this setup are computationally solved by so called *planners*, at which we are going to look at later. For now it is important to see how they attempt to solve an HTN planning problem. The first approach is called *plan-based HTN planning*, and we are going to give just a brief overview of it and give a complete definition of the second technique, the one used by our planner of choice, *state-based HTN planning* (reference to papers).

With plan-based planning, given an HTN planning problem P, the initial task network tn_0 is repeatedly decomposed until a primitive task network is reached, that is a network of only primitive tasks. Decomposition is refining a compound task in a further task network by a method, the formal

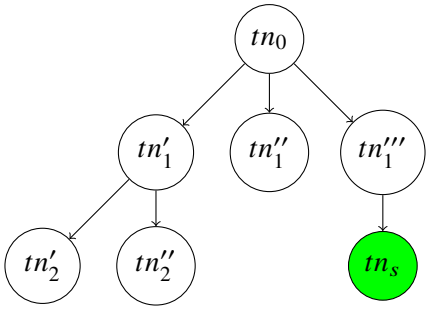

Figure 2.1: An example graph of plan-based searching space for an HTN planning problem P.

definition for which will be introduced with state-based planning. We will now give an illustrative example of how the search space during decomposition in plan-based planning could look like with the following graph.

In Figure 2.1, tn_0 is the initial task network and it has three outgoing edges, meaning that it can be decomposed by three different methods - $tn_0 \rightarrow tn'_1, tn''_1, tn'''_1$. One of the resulting task networks, tn''_1 cannot be further refined and it includes at least one compound task, which means that it cannot be a solution to the P. tn'_1 can be decomposed in tn'_2 and tn''_2, but they also contain a compound task and consequently are not a solution as well. The path $tn_0 \rightarrow tn''_1 \rightarrow tn_s$ leads to the task network of only primitive tasks, tn_s, a solution to P.

2.3 State-based HTN planning

This space structure is a subset of the state space [Geo15]. Here task decomposition restricts explicitly described states. We begin in s_0 with an empty plan, but instead of accomplishing a goal state, the search is for a state that will accomplish tn_0. If a task from the task network is primitive, it is executed and the search continues into a successor state. If it compound, we add its decomposition to the task network, but remain in the same state. If a solution is found, it is equivalent to a sequence of totally ordered primitive tasks, the plan. A task network is in this context specifically:

Definition 2.3.1 (Task network, state-based)
A task network $(T, \prec)$, where T is a finite set of tasks, and $\prec$ is a strict partial order on T (irreflexive, transitive, assymetric).

A task network tn does not allow multiple occurrences of a same task in the ordering expressed by $\prec$. We now will formally define what a decomposition of tn by a method m is.

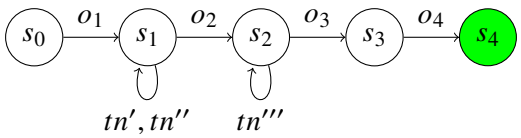

Figure 2.2: An example graph of state-based searching space for an HTN planning problem P.

Definition 2.3.2 (Decomposition)
Let P be an HTN planning problem, s a state in P, m an applicable method in s and $tn_c = (T_c, \prec_c)$ be a task network. Method m decomposes tn_c into a new task network tn_n by replacing compound task t, written $tn_c \xrightarrow[s,t,m]{D} tn_n$ if and only if $t \in T_c$ and $t = c(m)$, that is t is a compound task in T_c and m is a decomposition method for t, and

- $tn_n := ((T_c \setminus \{t\}) \cup T_m, \prec_c \cup \prec_m \cup \prec_D)$, *where*
- $\prec_D := \{(t_1, t_2) \in T_c \times T_m \mid (t_1, t) \in \prec_c\} \cup \{(t_1, t_2) \in T_m \times T_c \mid (t, t_2) \in \prec_c\}$.

The new task network tn_n contains the original tasks apart from t, and replaces t with the tasks defined in the task network T_m of the method m. The partial order of that newly emerged task network is the old order of $\prec_c$ between the old tasks, the order of $\prec_m$ between the new tasks, and $\prec_D$, which preserves the order of the new task to the old ones, in relation to how they were order with t. A solution to P is then:

Definition 2.3.3 (Solution)
Let P be an HTN planning problem. The sequence $o_1, \ldots, o_n$ is a solution to P, if and only if there exists a task $t \in T_0$, such that $tn_0 = (T_0, \prec_0), \forall t' \in T_0 : (t, t') \in \prec_0$, that is, t precedes all tasks in the partial order of tn_0, and

- *t is primitive and applicable in s_0 and the sequence $o_2, \ldots, o_n$ is a solution to P in which the task network is $tn_0 \setminus \{o_1\}$ and the state is $s_0[o_1]$*

or

- *t is compound and there is a task decomposition in s_0 such that the sequence $o_1, \ldots, o_n$ is a solution to P after $tn_0 \xrightarrow{D} tn'$.*

Here, the sequence $o_1, \ldots, o_n$ denotes a sequence of primitive tasks, which are equivalent to operators from Definition 2.2.2, as already discussed. Going back to the graph representation of the search space, the transitions now lead to a different vertex only if the decomposition is actually a primitive task being applied to the current state and consequently changing it. Vertexes are states of the problem and edges are either a compound task decomposition, thus a self-transition, or a primitive task leading to a different vertex, that is state. This clarifies the term for this approach, state-based planning. In Figure 2.2, the search begins in s_0. The primitive task o_1 can be directly executed and leads to the successor state, s_1. In s_1, we have to apply tn' and tn'' and by that decomposition, o_2 is now the first task in the partial order. We apply o_2, this modifies the state to s_2, in which we further decompose the task network by tn''' and finally execute $\xrightarrow{o_3} s_3 \xrightarrow{o_4} s_4$. The sequence o_1, o_2, o_3, o_4 is a solution to P, and only a transition marked with an o_i changes the state.

2.4 Planning language

Now it is finally time to look at how we can apply the formal definitions in a computational manner. We mentioned that planning is computationally solved by so called *planners*. The most common standard input for such planners is the Planning Domain Definition Language (PDDL), first introduced in [AHK+98] in 1998.

2.4.1 Planning Domain Definition Language (PDDL)

PDDL addresses the lack of common input for planners, which complicates the evaluation and comparison of their performance. At its core, it is an action-centred language, based on the STRIPS formulation of planning problems (Definition 2.1.3). It is intended to express the physics of a domain - its predicates, the possible actions in the domain, structure of compound actions, their preconditions and their effects [AHK+98]. Furthermore, we can encode a type structure for the objects in a domain and constrain the parameters of actions and predicates to be of a certain type. Separated from the domain behaviour, the problem definition is the description of specific objects, initial conditions and a goal description. The problem is in this sense an instance of the the domain, which can then be given to a planner. The objective of the planner is to find a solution to the problem, reasoning with the corresponding domain.

Since we want to include temporal observations in our environment, we are interested in version 2.1 of PDDL [FL03], which introduces *numeric fluents* and *durative actions*. This allows for the expression of temporal planning domains. The referenced papers provide a full formal definition of the language, and we will give an illustrative example Listing 2.1 of a domain and a problem, borrowed from [FL03].

As it can be seen, the syntax of PDDL is a Lisp-like list of parenthesised expressions. The *jug-pouring* domain models the action *pour* from the famous jug-and-water problem - pouring the water from the first jug into the second, given that the free space in the second one is large enough to hold it. Its effect is an appropriate update to the numeric fluent *amount* of both jugs, using the special keywords *assign* and *increase*. The problem instance, *jug-pouring-example*, instantiates two jugs, *jug1* and *jug2*, with a capacity of 250 and 500, respectively. We can see that the numeric expressions in PDDL are unitless. The initial amount of *jug1* is 0 and that of *jug2* is 250. The goal is an empty *jug2*, and for that the planner just has to execute action *pour(jug1,jug2)*, as its precondition is true in the initial state and its effect directly achieves the desired goal state.

PDDL itself is not capable of capturing the additional complexity of HTN planning. We cannot encode compound task decomposition and thus impose hierarchical relationship between the tasks with a plain PDDL domain, and therefore we need an extra specification, which builds on top and fills the missing gaps.

2.4.2 PDDL adaptations for HTN planning

The differences between planners that use hierarchical task decomposition are deeper, since domain description includes more than just domain behaviour (as already mentioned, the power expressiveness is controversial amongst the planning community). PDDL therefore does not attempt to propose a standardisation for this part of the language [FL03].

By exploring the relevant scientific literature, it is evident that hierarchical planning truly lacks a common input language, even when the planning systems are restricted to the best-known and most basic formalism, namely HTN planning [HBB+20]. Taking this into consideration, we are going to focus on the input language Hierarchical Plan Definition Language (HPDL) [Geo13] for our planner of choice, SIADEX [FCGP06], a planner capable of handling temporal hierarchical domains.

The work of Höller et al. [HBB+20] proposes a standardized language for hierarchical planning problems - Hierarchical Domain Definition Language (HDDL). Domains and problems in HDDL can be translated into HPDL by parsers like pandaPIparser [BHS+20]. HPDL itself embodies the syntax of the already presented PDDL2.1 and offers a few additional useful constructs, which are specific to the SIADEX planner, hence the aforementioned differences between hierarchical planners.

The essential construct of hierarchical planning, compound tasks and their decomposition methods, are mapped in HPDL the following way - Listing 2.2. A compound task consists of a list of its parameters and *n* decomposition methods *(:method ...)*, and the parameters can be used in each of them. Each method also has its own precondition, which determines whether the method can be applied. SIADEX attempts to apply them in the order that they are defined, that means if method_1's precondition fails, the precondition of method_2 is tested, etc. Of course, if none of the preconditions evaluates to true, the compound task itself is not applicable in the current state. Section *:tasks (...)* indicates the decomposition scheme of the corresponding method, with the possibility to define a variety of orders on the newly added tasks, as will be discussed later. The tasks in the decomposition scheme can either be a task, a task with temporal constraints *(<temp-constraints> (<name> <var-typed-list>))* or an inference task *(:inline...)*. The latter two are specific to SIADEX, the following points present these exclusive features of HPDL [FP20], and they also have a big influence for our choice.

- Temporal constraints over the start and end point, as well as the duration of a task at any level of the task hierarchy. Any subaction in a method has three special variables *?start, ?end* and *?dur* and constraints on their value can be posted by using logical expressions.
- Timed initial literals, as `(between "8:00:00“ and "20:00:00“ and every "24:00:00"(daytime))` - this represents that the literal `(daytime)` is true between a time point fixed at "8:00änd a another one fixed at "20:00", repeated every 24 hours.
- Inference tasks *:inline <precondition> <effect>* where *<precondition>* and *<effect>* are usual logical expressions for preconditions and effects of PDDL actions. They may be used to assert or retract literals into the current state or to capture information from it.
- Binding of a variable with the evaluation of an expression, using the special predicate *bind <var> <expression>*.

Listing 2.1 Domain and problem for pouring water between jugs, adapted from [McD00] to PDDL2.1

```
(define (domain jug-pouring)
    (:requirements :typing :fluents)
    (:types jug)
    (:functions
        (amount ?j - jug)
        (capacity ?j - jug)
    )
    (:action pour
        :parameters (?jug1 ?jug2 - jug)
        :precondition (>= (- (capacity ?jug2) (amount ?jug2)) (amount ?jug1))
        :effect (and (assign (amount ?jug1) 0)
                     (increase (amount ?jug2) (amount ?jug1)))
    )
)

(define (problem jug-pouring-example)
    (:domain jug-pouring)
    (:objects jug1 jug2 - jug)
    (:init
        (= (capacity jug1) 250)
        (= (capacity jug2) 500)
        (= (amount jug1) 0)
        (= (amount jug2) 250)
    )
    (:goal = (amount jug2) 0)
)
```

We are going to see an example for each of these features in our implementation in Chapter 5. It is important to note that it is exactly such features for well-structured domain knowledge that are controversial and problematic. They can be easily used to guide the search and thus break with the philosophy of PDDL to specify a model that does not include advice [HBB+20].

2.5 SIADEX

The last background information needed for our setup is the HTN planner itself, SIADEX, or HPDL-Planner. Efficient and expressive handling of time is still a challenge for most HTN planners [CFGP06], and since our domain of interest requires temporal information to be precisely defined, SIADEX seems as a very logical choice. It integrates innovative techniques to efficiently handle temporal knowledge so as to allow a fast response of the planning system, also a requirement of big priority in ubiquitous computing.

Algorithm 2.1 shows the main algorithm of SIADEX. It takes the set of tasks to be achieved, explores the space of possible decompositions and replaces a given task by its component activities, until the set of tasks is transformed into a set of primitive actions - the plan. A crucial feature of

Listing 2.2 An HPDL task [FP20]

```
(:task <name> :parameters <var-typed-list>
    (:method <name>
     :precondition <prec>
     :tasks (
        (<name> <var-typed-list>)
        (<temp-constraints> (<name> <var-typed-list>))
        (:inline <condition> <consequence>)
     )
    )
    (:method ...)
    ...
)
```

SIADEX is the **Propagate-Temporal-Constraints(Π)** call in line 8. It assures that the temporal constraints of the primitive actions that are stored in Π do not contradict each other, that Π can be temporally executed. An explicit description of how this is implemented is given in [CFGP06].

In this sense, another very convenient property of SIADEX, which is expressed in HPDL, is its possibility for expressing a variety of orderings on the subtasks in the decomposition scheme of a method, as in the *:tasks (...)* subsection in Listing 2.2.

- $(T1, T2)$ is a **sequence**. Tasks defined in a sequence must be executed exactly in the same order, that is, first $T1$ and then $T2$. Any subtask of $T1$ and $T2$ inherit this order as well.
- $[T1, T2]$ are **unordered**, that is, either $T1$ or $T2$ could be executed first, or they could run in parallel.
- $< T1, T2 >$ is a **permutation** and must be executed in any of the total orders given by any of their permutations, in this case either $T1$ and then $T2$ or vice versa. Subtasks inherit these relationships.

Our implementation takes advantage of these partial orders and we are going to see in Chapter 5 how this allows us to recognise activities at different locations in parallel temporal branches.

Algorithm 2.1 A rough outline for the HTN planning algorithm of SIADEX [CFGP06]

Input: $T := tn_0$, the initial task network; $S := s_0$, the initial state
Output: Π, the plan of primitive actions

1: $\Pi \leftarrow \emptyset$
2: **while** $T \neq \emptyset$ **do**
3: Extract a task t from T
4: **if** t is a primitive action **then**
5: **if** *pre(t)* is satisfied in S **then**
6: Apply t to S, $S = (S \cup eff^{+}(t)) \setminus eff^{-}(t)$
7: Insert t into the plan, $\Pi = \Pi + \{t\}$
8: **Propagate-Temporal-Constraints**(Π)
9: **else**
10: **FAIL** // no plan can be created → no output
11: **end if**
12: **else** // t is a compound task
13: **if** there is no decomposition method m with $c(m) = t$ **then**
14: **FAIL**
15: **end if**
16: Choose one method m, such that $c(m) = t$ and *pre(m)* is applicable in S, and map t into its set of subtasks $\{t_1, \ldots, t_n\}$
17: Insert $\{t_1, \ldots, t_n\}$ into T
18: **end if**
19: **end while**
20: return Π

3 State of the Art

Now that we have the needed basic theoretical foundation for HTN planning, it is time to look at relevant approaches to recognising activities in the literature. The successful automatization of this task would be of great benefit to various domains, from supporting the monitoring of elders, through optimizing the energy consumption in offices, to extending the comfort of the modern home.

In general, we are still not quite close to a framework that is widely accepted as the best or right way to recognise activities, to the system that yields undeniably the most accurate results. Experiments from the literature vary in their assumptions and consequently in the applied methods to prove their assumptions. Nevertheless, attempts in activity recognition are getting more and more complex and refined, achieving better results and filling the gaps of earlier researches.

There are two broad categories of sensor based activity recognition approaches - data and knowledge-driven [INIT18]. data-driven approaches use machine learning and statistical methods which involve discovering data patterns to make activity inferences. Their advantage is that they can deal with incomplete or noisy data, however, they lack the expressiveness to represent activities. Furthermore, they need large-scale labelled datasets to train and learn their data model, which produces scalability problems for practical deployments.

Our implementation is solely based on hierarchical planning, but an extension to it with data driven techniques could increase without a doubt its performance, as incorrect sensor data has to be expected in a real world situation. Knowledge-driven approaches, on the other hand, specify the conceptual structure of activities and their interrelationships, represented in an ontological model. This resonates with the concept of hierarchical structures and we are going to see how we can apply it to existing ontological approaches.

3.1 Activity recognition by sequence of objects

A work of importance in the community is that of Philipose et al. [PFP+04], which uses a clever combination of data and knowledge-driven approaches to infer activities in the context of elderly care. It tests 14 ADLs, which is 11 more than any system had previously attempted, and for 9 of the activities it is the first ever attempt. Their key observation is that the sequence of objects a person interacts provides a solid basis for identifying both the current activity of the person and its quality of execution. They tag objects with radio-frequency-identification (RFID) tags, as they can be unobtrusively attached even to small objects. The person whose activities are to be recognised is equipped with a glove-based RFID reader, which can identify the tagged object, hence the sequence of touched objects to be processed. The achieved results in the experiments are impressive, as 88% of the time a newly occurred activity was accurately detected (this does not include the false positives). Their reasoning system, *PROACT*, represents activities as linear sequences of activity stages. For example, they model the making of tea as a three-stage activity: (1) boil the water; (2)

steep the tea in the water; (3) flavour the tea with milk, sugar, or lemon. Here, we see a fist major drawback of this model, namely its linearity. The authors point out that users can perform ADLs in various ways, and models must adapt to this variety. It is often tedious to extend systems to cope with this, but HPDL and SIADEX offer us a very convenient way to express multiple ways a user can make a tea:

- (boil water, steep tea in water, flavour tea) - we expect to see the person, who is making a tea, to perform these subactivites in exactly this order.
- [boil water, steep tea in water, flavour tea] - now the user can first steep tea in water, then boil the water and flavour the tea, he can perform them concurrently or in any other order.

A strict linear order on making tea might still yield a high percentage of its recognition, as it is a task that is usually performed in this linear sequence. However, when we go up in the hierarchy of activities and respectively in a higher level of abstraction, a strict order is bound to limit the capabilities of the model.

Another advantage of HTN planning is the possibility the define multiple ways the user can perform a given activity - for example, flavouring the tea. Here we can use a method (Definition 2.2.4) and its respective decomposition (Definition 2.3.2) schemes. Flavouring the tea is a compound task, and it has three methods - flavour with milk, flavour with sugar and flavour with lemon. If the user touches the milk, then the preconditions of flavour with milk are fulfiled and the planner chooses to apply this method decomposition for the compound task flavouring the tea. It is a very natural representation, that follows the paradigm of knowledge-based inferences that are easily understood by the end user [CHN+12]. In contrast, in *PROACT* each stage is annotated with the objects it involves and the probability of their involvement, as how likely it is for the user to flavour his tea with the respective ingredient. These probabilities combine sensor error, model error, and model generality. Afterwards, the probabilistic engine converts the activity models into dynamic Bayesian networks to infer activities. It is far less intuitive to the human mind, but also captures complexity of the real world that cannot be encoded in pure HTN planning.

We can also see how a hierarchical structure can deal with another issue in *PROACT*:

[Activities with starting points posed a more subtle problem. For instance, activities for personal appearance (ADL 1), oral hygiene (2), toileting (3), and washing (4) all begin with entering the bath- room and possibly turning on the light. Our models replicate the nodes for these subactivities for each of the four activities. When PROACT detects someone entering the bathroom and turning on the light, each of these four activities is equally likely. When the system detects disambiguating objects (such as a tooth- brush and toothpaste), the inference engine concludes that oral hygiene is the correct activity. If the user then uses the toilet without leaving the bathroom first, the inference engine fails to detect the second activity. To solve this problem, we might need to consider more nuanced representations of activities.]

In Figure 3.1 compound task are marked in red, and primitive in blue. ***use bathroom*** is decomposed by two methods, which are also compound tasks. Each of the tasks that represent an activity can have a precondition with the objects that were last touched, as for example ***oral hygiene*** would require *touched toothbrush* and *touched toothpaste*. If the recursive ***multiple bathroom activities*** fails, then ***bathroom activity*** is tested, and this ensures that any number of iterations of bathroom activities can be represented.

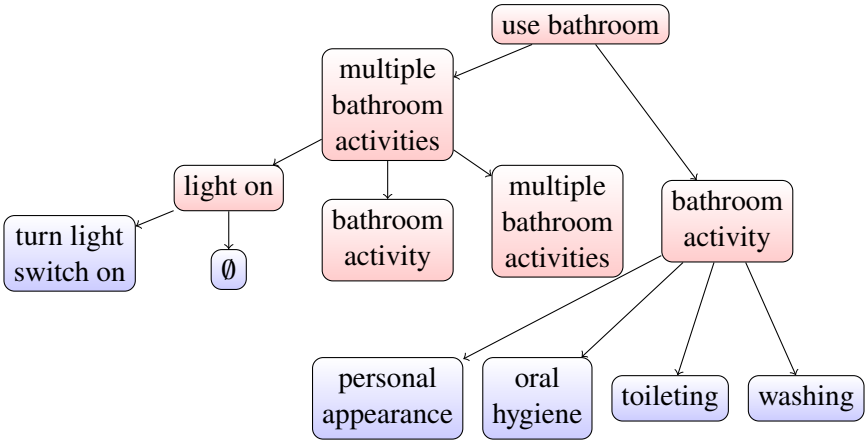

Figure 3.1: Hierarchy for recursive bathroom activities

A further drawback pointed out by the authors is the fixed mean duration of an activity, which becomes problematic when an activity like boiling water is taking place, as it has a considerably longer duration than most of the other activities. HPDL allows us to pose temporal constraints over the start, the end, and the duration of a task at any level, as pointed out in Section 2.4.2. Now a fixed mean duration is not needed and we can assign easily a duration of five minutes to boiling, as opposed to probably 30 seconds for washing hands.

3.2 Activity recognition using hierarchical plans

Another approach using RFID sensors and a RFID reader to identify them is presented in [NBW]. Here we can see that the suggestion by Philipose et al. [PFP+04] that their method using glove-based RFID reader will be adapted to a much more portable device is indeed plausible, as the reader in this experiment is the size of a match box and is worn on the finger. Their tiered framework of interpreting sensor data when monitoring ADLs is innovative and improves on limitations of [PFP+04]. In order to compensate for missing sensor events on the lower level, they establish Task Associated Sensor Events as an input to a segmentation algorithm, based on a statistical model. The end result is the most likely task which is currently active. This is further fed to the higher level activity recognition process, defined by a hierarchical plan representation language, Asbru. The authors perform different experiments, using distinctive and non-distinctive sensors (meaning sensors that specifically relate to a given task and such that do not), and ADLs are executed in either a predefined order, a random one or even interleaved. For this variety, the authors achieve a quite impressive average detection rate of 86,3% after all experiments.

A part of the reason for this result can be attributed to the hierarchical representation of activities, as opposed to the strict linear sequences in [PFP+04]. The approach was able to perform well even without distinctive sensors because of the planning capability of the planning language - it can deal with tasks which occur in any order or are missing, as long as few tasks which are associated with a given ADL have occurred. It can represent all of them as plans to be recognised. We argue that HPDL is even better suited for the hierarchical representation of activities. We can see that the

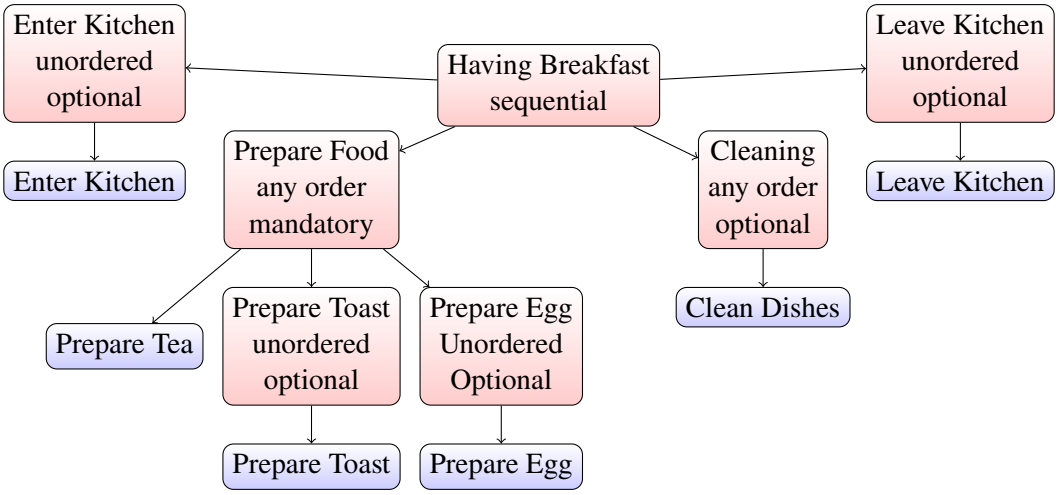

Figure 3.2: Hierarchical model of ADL Having Breakfast as in [NBW]

hierarchical representation of ***Having Breakfast*** in Figure 3.2 from [NBW] is similar to Figure 3.1. Furthermore, we can easily map the sequence order of the subactivities in Figure 3.2 to HPDL, as for example:

Having Breakfast (Enter Kitchen, Prepare Food < Prepare Tea, Prepare Toast, Prepare Egg > , Cleaning, Leave Kitchen)

With HTN planning, we do not need to replicate nodes in the hierarchy for optional activities, as for ***Enter Kitchen*** $\rightarrow$ ***Enter Kitchen***. The decomposition methods allow us to define empty task decomposition, indicating that the compound task can also just be skipped, as for ***light on*** in Figure 3.1.

A limitation of the recognition system in [NBW] and a suggestion for its future enhancement is that it does not integrate time information in its process, as for task durations or context for the time of the day, as daytime or nighttime. This is again easily encoded in HPDL, as described in Section 2.4.2. For example, by using the `daytime` predicate, we can infer whether having a meal is actually a breakfast, lunch or dinner.

Another work using hierarchical plans is that of [FP20]. The recognition for activities here is for a different domain, namely recognising driving activity according to the Hours of Service regulation, beginning from an event log of the driver's activities. The event log should be parsed in the top level driving sequences, defined by the regulation, as normal or extended daily driving sequence, weekly driving. This makes evident if the driver is committing violations to the regulation and potentially saves sanction costs for the transport company.

The nature of the regulation is hierarchical and the temporal context is of big importance for parsing the subsequences, therefore the authors use HPDL for mapping the observations from the event log to a HTN problem, and SIADEX for solving it. They further translate the Hour of Service regulation rules by the means of an attribute grammar into a temporal HTN domain, capable of identifying and parsing the event log. The end result is a labeled event log that is easily interpretable by experts.

It is a novel temporal HTN approach to plan and goal recognition where the plans observed are temporal sequences of events. In this context, it represents a first attempt to model behaviour as a planning domain, including temporal and numerical information.

Since our the idea behind our approach is to recognise activities by their hierarchical relationships and temporal context, our approach is inspired by [FP20] and more specifically their planning techniques, HPDL and SIADEX. We are going to see in Chapter 4 how we apply ideas and design choices from [FP20] to our design. However, it is important to note that our environment and setup differentiate with theirs.

Firstly, detecting if the activity of a driver complies with the regulation can be seen as a plan that is or is not generated by a regulatory model. We do not have a formal model that specifies human activities, as they can be performed in any random way, thus introducing an additional level of complexity to the problem. Moreover, the recognition of driving activity assumes full observability, noiseless and complete event sequences, and no ambiguity - assumptions that do not hold in the domain of recognising general ADLs in whichever environment. For example, even when there is a 'gap' in the event log of a driver, it can be directly interpreted as a pause or idle activity with a duration equal to the size of that gap. In the case of a missing sensor reading in the home environment, probabilistic data-driven techniques like the segmentation of sensor events from [NBW] have to be deployed to compensate for the incomplete data, as ambiguity now has to be expected.

3.3 Ontology-based activity recognition

Nguyen et al. [NRA13] explore office multiple-user multiple-locations activity recognition, like working with PC or having a meeting, in the context of building energy and comfort management system. The goal is to infer what is happening in each of the monitored areas (working place, meeting room and coffee corner) so that appliances can be controlled accordingly in order to save energy. In other words, activity recognition is a major research issue to realise intelligent pervasive environments, a big challenge in the vision towards ubiquitous computing.

Office activities, which directly influence energy saving in buildings, are limited and consequently can be predefined and represented as domain knowledge by experts. The authors use ontological modeling, representation and reasoning on information, gathered by a wireless network of low-cost and easy-to-deploy sensors. While earlier research focused on monitoring and analysis of invasive technologies like cameras or wearable tags, more recent research has moved towards a network of multiple miniature dense sensors embedded within environments [CK11]. Therefore, our solution is also based on the premise that the data is coming from binary sensors or such that report a numerical value, and we do not consider invasive technologies. Moreover, processing such complex data structures is not really feasible by the means of HTN planning.

In [CK11], activities, artefacts and locations ontologies are further refined by semantic relationships and constraints between these ontologies, a rough overview is shown in Figure 3.3. A specific set of `Artefacts liesIn` in a given `ActivityArea`. The `Activity happensIn` an `Activity Area`, and `hasDetected Artefacts`. In the other direction, at any given moment, an `ActivityArea isRunning` one particular `Activity`.

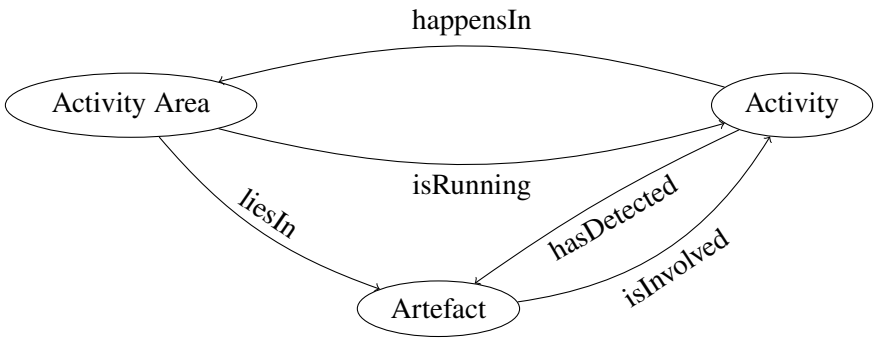

Figure 3.3: Ontology for recognising activities in offices [NRA13]

The work of Georgievski et al. [GNA13] describes how the reasoning algorithm works, as they use the same ontology Figure 3.3. First, a time interval for reading the vector of sensor readings is set. This interval has to be determined in regard to how often we want to update the state of the environment. In the domain of energy saving in offices this would be a window that allows for turning on/off appliances as soon as there is a need to. For example, in [NRA13], this interval is set to every second. We are going to see how HTN planning allows us to abstract from a specific time frame, and instead each sensor can push its readings independently.

For every time interval, for every `Activity Area` entity, all of its `Artefacts` are retrieved by the relation `liesIn`. As the state of each `Artefact` is known, now the currently active `Activity` can be determined by using the interrelations between the two ontologies. For example, if in the *Working Room* `Activity Area`, the electricity measuring plugs for a monitor and a computer report a state *On*, and a pressure sensor attached to a chair indicates that somebody is sitting on it currently, it can be determined that the `Activity` that is `runningIn` *Working Room* is *Working with PC*.

The ontology-based approach shows very promising results for the domain of energy saving in buildings. Although [NRA13] is the first work that investigates and proposes such a solution for area-based activity recognition, the performed experiments for three office activity areas for two persons achieve an average accuracy of more than 92% while recognising six distinct activities. Georgievski et al. [GNA13] use this ontology model and further deploy HTN planning to construct a plan of actions to control the state of two workstations and ceiling lamps at three locations - HTN planning here uses the already recognised activities as input to control the office space. Although the accuracy of around 80% of recognised activities in these experiments is considerably lower, it satisfies the requirements of energy saving in buildings. The experiments were performed on three different days and the energy saving potential was in the order of 70%.

Besides the typical problems with knowledge-driven approaches, limitations of these two experiments, as they are based on a network of simple sensors, are a partial observability of the environment and potentially corrupt and imprecise sensor data[NRA13]. A small loss in detail in recognising activity can be tolerated for the sake of cost and ease-of-use in the domain of energy saving in buildings [NA13], which is evident from the results in [GNA13].

3.4 Hybrid approaches

In the more recent literature, a rise in the combination of both data- and knowledge-driven approaches can be observed. The resulting systems use data reasoning on the lower level of abstraction, as interpreting the sequences of sensor events. Different ontological models are then used to represent the structure of activities and to match sequences of events to activities. In fact, such a hybrid approach is the already discussed one of Naeem et al. [NBW].

Another one is the activity recognition system USMART [YSD14]. The authors segment the sensor events using the semantic similarity between the fired sensors. Based on previously modelled ontologic activity models, they define the sufficient conditions for a sensor sequence to be mapped to an activity by semantic reasoning. In order to distinguish an activity from others, the system needs at least one unique sensor event to describe it.

The HARS system [AA18] arguably improves the framework of USMART. In an unlabelled dataset produced after monitoring of the activity of a person, data-mining techniques are used to find the most frequent action patterns in it. The difference with USMART is that sensor readings are not segmented according to their semantic similarity. Here, the most frequent patterns executed by the monitored person are searched, which makes the framework more generic and better adapted to sensor noise. For example, their algorithm can also identify patterns that do not belong to any activity, and therefore can label incorrect sensor events. In order to infer which activities are being performed in a given action pattern, expert activity models (knowledge-based computational models for a given activity) are used in a pattern-model matching algorithm. This expert model includes minimal generic information about the activity, which makes it relevant for varying activity executions.

A more complex hybrid approach is that of Ihianle et al. [INIT18]. It consists of two components - the context description module and the ontology module. Object use for specific routine activities as activity-object use distributions and activity context descriptors are discovered by the context description module. An activity ontology is developed using the discovered object use for the respective activities as ontology concepts in the ontology module. Activity recognition is achieved by observed object use query on the activity ontology for the relevant activity situations. This framework is indeed sophisticated, but introduces a great modelling effort for adaptations to a new scenario in real world deployments.

All [AA18; INIT18; YSD14] report results on the same dataset, and we are going to compare them with our framework in Chapter 6. In the next section, we are going to investigate the design of our solution, taking inspiration and extending ideas from the presented approaches.

4 Design of Hierarchical Planning Activity Recognition

An overview of the architecture of HPAR is shown in Figure 4.1. We can see that the main source of information comes from the log of sensor events, which we translate into a HTN planning problem. How this log is created is irrelevant to our system, and we describe in Section 4.1 how we translate the sensor readings into predicates for the problem. Then the planner searches for a solution of this problem, using the knowledge from the domain model. The design of the domain is presented in the current chapter, and the concrete implementations of the domain and problem follow in Chapter 5. An example output solution, produced by the planner SIADEX is listed in Listing 5.8.

The first challenge in designing the framework of HPAR is modelling sensor data, including contextual data and important features of the reading, and transforming it into the respective data format to be processed.

4.1 Sensor Data

We base our model on low-level sensor data, as in [NRA13], that is we do not consider technologies like cameras that would require reasoning beyond a simple interpretation of the reported value, besides the already listed privacy issues that come with using invasive technologies to monitor people.

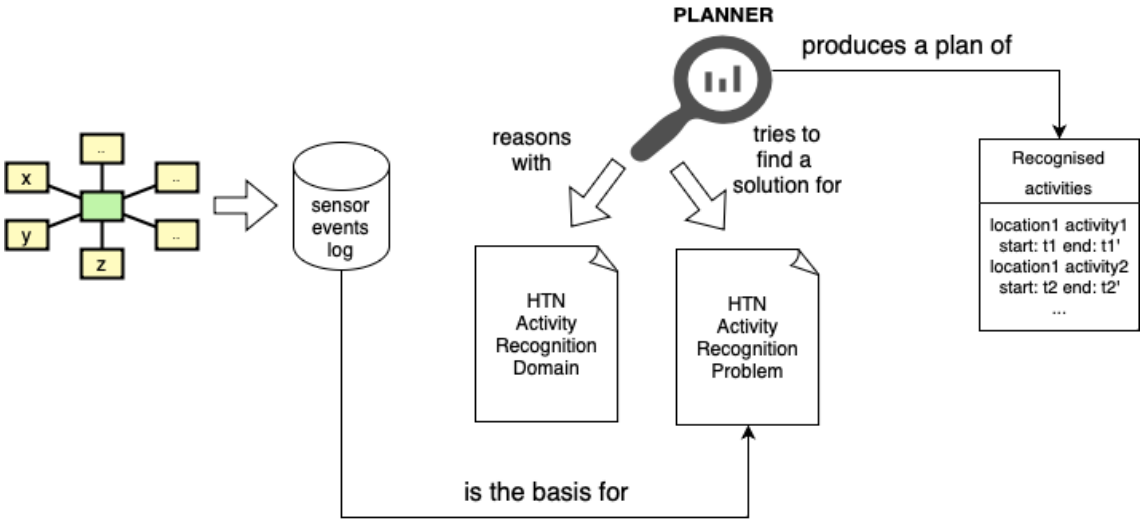

Figure 4.1: Overview of the system architecture

As we want to recognise activities at different locations, first we need to provide background for the location of each sensor. This is simply achieved by defining a predicate in the initial state of the HTN planning problem, Definition 2.2.5. For that, we have to create a variable for each sensor and each location in our environment, Definition 2.2.1. For example, given that we have the locations `working room, coffee corner` with the sensors `PC electricity plug, monitor electricity plug, chair pressure sensor, infrared` and `coffee machine electricity plug, microwave electricity plug, infrared`, respectively, we can represent them as follows:

$$\{\texttt{working room, coffee corner}\} \in V$$
$$\{\texttt{PC, monitor, chair working room, infrared working room}\} \in V$$
$$\{\texttt{coffee machine, microwave, infrared coffee corner}\} \in V$$
$$\{\texttt{sensor at(PC, working room), sensor at(monitor, working room),}$$
$$\texttt{...sensor at(infrared coffee corner, coffee corner)}\} \in P$$

The only assumption about the data reported from a given sensor we want to make, is that it reports either a *binary* value (as in *On/Off* state) or a *numerical* value (as a *100 N* for a pressure sensor), which can be again mapped to an *On/Off state*. Whether a sensor is binary or numeric can be directly encoded in the type hierarchy of the PDDL Listing 2.1, respectively HPDL domain:

```
(:types binary numeric - sensor PC monitor ... - binary chair infrared - numeric)
```

Assumption like this is loose in the sense that it provides the network of sensors with freedom in regard to how they report the data. Each sensor can push data to its own rhythm, regardless of whether it is a fix-sized window as in [NRA13] or a dynamic window approach as in [LOB11], that reacts on changes in the sensor state. Instead, we can abstract from a specific time frame and just represent each new reading as a predicate, including the temporal point of the reading, in the HTN planning problem, Definition 2.2.5. This design is generic enough to be applied to both a network of sensors, as in [NRA13], and a collection of objects, tagged with RFID sensors, as in [PFP+04]. Actually, it can be a mixture of both, the organisation and types of sensors used is irrelevant, as long as each sensor fulfils these requirements.

Indeed, this is a very high-level restriction on the system, as it does not really indicate the type of network the sensors have to form, or the technologies that can be deployed to realise this network. However, we see this as a big advantage to our solution, as it indicates generality that many other solutions lack.

For example, for gathering sensor data, we can easily use the open source lightweight *publish/subscribe* protocol for the Internet of Things (IoT) - MQTT [MQT]. The clients in this sense would be the single sensors, sending data to different topics on their own terms, and the message broker can be realised through open sources brokers like RabbitMQ [VMW] and Mosquitto [Ecl]. For the translation to predicates, we need the data from the broker in the sequence that it receives it. The essential information we need is the value of the sensor reading, the temporal point of the reading and a further index to represent at which position in the sequence of readings this particular reading is. An example of this transformation is shown in Figure 4.2. We do not need to explicitly consider the specific time frames that the different topics use and can just translate each topic message to predicates for its values and a predicate for the timestamp. Topic y pushes at timestamp

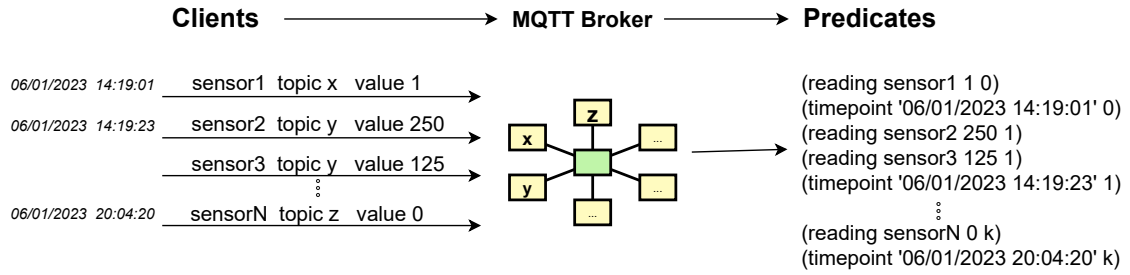

Figure 4.2: Mapping of MQTT messages to predicates

`06/01/2023 14:19:23` values 250 and 125 for its two sensors `sensor2` and `sensor3`, respectively. As the two sensors belong to the same topic *y* and correspondingly push values at the same times, their `(reading...2)` predicate is associated with the same index 2.

It is important to note that the index term of the predicate goes against the principles of PDDL, as it guides the search of the planner in a very strict sequential order. However, if our goal is to recognise temporal activities, it is obviously essential to make the planner process the predicates in this strict order, as only then we can properly reason about the current activity. In other words, if we exclude the index and the planner chooses which predicate to read next in a random order, let us say `(reading PC 1)` for timestamp t, it concludes that the PC is on at time t. It then reads `(reading monitor 1)` at t' and concludes that the user is now working with PC. As the readings are now not ordered, it could be that the PC has been switched off between t and t', it could be that t' is actually before t. Reading the sequence of events in the right order is of biggest importance to accurate recognition. The index approach is taken from [FP20], as they use it to parse the log even in a linear order.

We use historical data, that is our algorithm is applied *offline*, once the data has been gathered. However, as a potential future work improvement, the system can easily be extended to reason *on the fly*, processing the input instantly once it is pushed. Furthermore, the index would then be redundant, making the whole problem setup more generic and close to the philosophy of planning.

4.2 Activity recognition

Our reasoning takes inspiration from the ontological model from [NRA13] and follows the philosophy from [PFP+04], namely that the sequence of objects a person uses is indicative of his current activity.

Listing 4.1 shows an intuitive overview of our approach to recognising activities. One certain activity is taking place at a given location for a specific timepoint. We can define a compound task `(recognise activities at different locations ...)` which is decomposed (Definition 2.3.2) in further compound tasks for each location in the set of locations, as `(recognise activities in working room ...)`. The purpose of the compound task for a single location is to go over the sequence of predicates and reason which activity is active at the given index and respectively timepoint, and eventually add a primitive task representing that activity, including temporal information about its start and end, once it is over and another activity is taking place. That primitive task is then added to the plan, the solution of the problem, Definition 2.3.3. When the planning process is

Listing 4.1 Rough description of the process of recognition of activities

```
(recognise activities at different locations)
    is decomposed to (recognise activities at location_i) for each location
        is decomposed to a sequence of primitive tasks annotated with start and end timepoints
        where each primitive task corresponds to a recognised activitity
            and each recognised activity corresponds to a set of involved-not involved sensors
            and each sensor is associated with location_i
```

over, the solution is constituted of primitive tasks for the recognised activities at each location. It is important to make sure to avoid temporal conflicts between the primitive tasks, as activities at the different locations are happening parallel to each other. Therefore, the relevant features of the planning languages have to be used when defining the compound tasks and their corresponding methods, as the inference tasks and temporal constraints of HPDL, described in Section 2.4.2, and its ordering of subtasks, described in Section 2.5.

As to recognising the activity itself, it is based on the state of the sensors at the location. Both binary and numeric sensors relate to an is involved/is not involved state (on/off state), and by mapping the different variations of states for sensors at a location to an activity, we can infer which one is active at this location at a timepoint. For example, having meeting is an activity happening only in the meeting room, and it involves an acoustic sensor detecting human voice, and two pressure sensors indicating that they are occupied. Both the acoustic and pressure sensor report a numeric value for the sound level (dB) and pressure (N), and a threshold value has to be set to indicate when exactly this value can be interpreted as detecting (human voice, a human sitting) or not detecting. Furthermore, we have to deal with the challenge of changing threshold values. Nguyen et al. [NRA13] report that, while for pressure or infrared sensors an absolute threshold value can be set, changes in sound level depend on offices and settings. Therefore, the threshold has to be adjusted through time. What logic is used to make this adjustments is out of the scope of our system, but we have to reflect the possible threshold changes to assure an accurate reasoning process.

4.3 Hierarchical representation of an activity

Now that we have a basic design for recognising plain activities, we can also represent more complex relationships between single activities, as trying to recognise the subactivities of an activity instead the activity as a whole.

A representation of the task washing hands and its component subactivities is shown in Figure 4.3 [BHP+06]. We can see three main characteristics that we have to be able to express - first, subactivities can be executed in different orders as different branches of the same node. Second, different branches might also include exclusive to the branch activities, as wetting hands is only sensible if we have turned the water on before using the soap. And third, different branches can again merge into a common node, as rinsing hands.

Hierarchical planning offers a very intuitive way of defining such relations in the decomposition scheme for a task. Let us see how we can declare such a scheme, using the different subtask sequences of HPDL, Section 2.5, where compound tasks are marked in red, and primitive in blue:

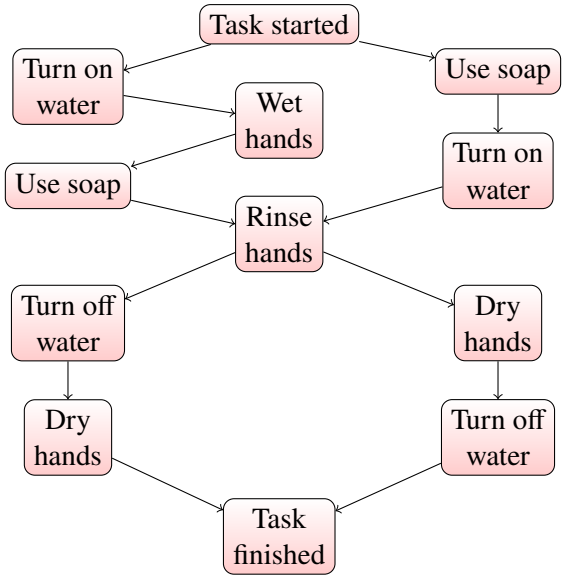

Figure 4.3: Hierarchical model of washing hands [BHP+06]

wash hands (soap hands$_1$, rinse hands, dry hands)
soap hands$_1$ (m_1: (wet hands), m_2: (soap hands$_2$))
wet hands (turn on water, wet hands, use soap)
soap hands$_2$ (use soap, turn on water)
dry hands < turn off water, dry hands >

Formally, each subtask () or <> sequence here represents a method. Only soap hands has two decomposition methods and therefore they are explicitly mentioned, whereas the other sequences represent the only method for the compound task.

Going back to the concept of activity recognition, we can now also apply it to hierarchical activities. For example, the planner is currently executing the task (recognise activities at bathroom) and attempts to apply its decomposition method wash hands at index k. This consists of sequentially recognising the primitive tasks of wash hands in any of its variants defined by the decomposition methods of wash hands, e.g. use soap, turn on water, rinse hands, turn off water, dry hands. If the planner successfully recognises the first few tasks, but then does not find the needed precondition of let us say rinse hands at index k', it does not create a gap in the sequence of events, as the planner would backtrack, removing the already recognised primitive tasks and their effects from the plan, and eventually start looking for another activity, starting at index k.

In the next section, we are going to investigate how this design can be technically realised.

5 Realisation of Hierarchical Planning Activity Recognition

For the technical implementation of our system, we first present the different technologies that constitute our system.

5.1 Technology stack

We do not consider any data-driven approaches to evaluate and classify sensor readings, as this is out of the scope of hierarchical planning. We also provide a very generic interface to define predicates for the sensor readings and therefore do not restrict the input to a specific network of sensors. The only details of importance are the sensor location, its type (this has to provided only once in the beginning) and threshold (if its output is a numeric value), its reading and timepoint of the reading. As discussed in Section 4.1, we can easily deploy a sensor network communicating through the MQTT protocol with a broker, or we can gather data from objects, tagged with RFID tags. From the point of view of our system, this is really not of importance, and we are going to present how we gathered data for evaluation of our system in Chapter 6.

What is essential is the model of how activities are recognised, that is which sensors are involved/-explicitly not involved in the recognition of particular (sub)activities and their hierarchical relations. As thoroughly discussed, for the computational representation of temporal HTN planning, we use HPDL as planning language and SIADEX as a planner. This is a reasonable choice since SIADEX is, to the best of our knowledge, the only publicly available temporal HTN planner and it uses HPDL as input language. HPDL can be translated from the more generic hierarchical planning language HDDL by parsers like pandaPIparser [BHS+20].

In other words, the specific technologies we use for the creation of the HPAR model, are HPDL and SIADEX. This model is represented through an HPDL domain, aimed at recognising a certain set of activities happening at again specific locations. The goal is to define the model in a way that allows to be easily understandable and extensible, as a given definition of it is inevitably mapped to a set of activities and obviously not all possible activities that can happen in our daily life. A problem instance of the domain relates to a data set gathered by the broker, an extract of the sensor readings through a time frame. The problem instance is parsed by SIADEX and the end output is a plan for the problem instance - the recognised activities at the different locations during the time frame.

Listing 5.1 A type hierarchy for activity recognition model

```
(:types
  sensor location activity - object
    binary numeric - sensor
    PC monitor projector coffee_machine microwave - binary
    chair infrared acoustic - numeric
    working_room meeting_room coffee_corner - location
)
(:constants
  absence presence working_with_pc working_wout_pc giving_presentation meeting having_coffee
)
```

5.2 Domain for recognising temporal activities

The activity recognition model is represented through HPDL domain knowledge. We have to define the types, predicates, functions (numeric fluents), tasks (compound tasks) and their (decomposition) methods, and the (primitive) actions of the domain.

5.2.1 Types

In the types section of the domain we have to define all main types of objects that the further subtypes inherit from - `sensor, location, activity`. A `sensor` can be `binary` or `numeric`. As an example, we show the type hierarchy for the ontology in [NRA13] - Listing 5.1.

We can see that activities are defined separately in the `(:constants...)` section. The reason for this is that we can now reference a specific activity directly by its `activity` constant instead of trying to infer the type of an `?activity` variable. The type of activity is always explicitly known when used in our domain (with a minor exception) and defining its type as a constant spares SIADEX a considerable amount of confusion when trying to find matching types for the terms of a predicate or task parameters.

5.2.2 Predicates

The next definition of the domain is the predicates section. Predicates apply to a specific type of objects, or to all objects. They are either true or false at any point in a plan and when not declared (in the problem instance) are assumed to be false, as in the closed world assumption (Section 2.1).

The sensor predicates must express their type, readings, location, and whether they are active. We need predicates to infer the type of locations, predicates to represent the current activity and its temporal information at a location, and predicates to represent the current index and the timepoints of the readings. For example, let us consider three different locations - working room, meeting room and coffee corner, then the respective predicates section would be as in Listing 5.2. We can see the predicate `(new_threshold...)`, which deals with the problem of changing thresholds, described in Section 4.2. It is bound to an index as the `(reading...)` predicate, because its place in

Listing 5.2 Predicates for the domain

```
(:predicates
    (is_binary ?s - binary)
    (is_numeric ?s - numeric)
    (new_threshold ?s - numeric ?value - number ?i - number)
    (sensor_at ?s - sensor ?l - location)
    (reading ?s - sensor ?value - number ?index - number)
    (sensor_active ?s - sensor)
    (exists_sensor_active ?l - location)

    (is_working_room ?l - working_room)
    (is_meeting_room ?l - meeting_room)
    (is_coffee_corner ?l - coffee_corner)

    (current_activity ?l - location ?a - activity)
    (start_activity ?l - location ?timepoint - number)

    (index ?i - number)
    (last_index ?i - number)
    (timepoint ?timepoint - number ?index - number)
)
```

the sequence of readings is just as important. When a sensor decides to change its threshold, the `(new_threshold...)` has to be inserted at the same or before the index of the readings that relate to this threshold.

We further complement some of the predicates with a corresponding `(:derived predicate)`, which indicates when `predicate` holds true. We have one derived predicate for both `is_binary` and `is_numeric`, which enables us to determine the type when the supertype sensor is passed as a parameter (as in the precondition of the methods in Listing 5.3). We declare

```
(:derived (exists_sensor_active ?l) (and (sensor_at ?s ?l) (sensor_active ?s)))
(:derived (index ?i) (bind (current_index)))
(:derived (last_index ?i) (bind ?i (last_index))),
```

the first of which tells us if there is any active sensor at a given location ?*l*. The other two use the special predicate *bind* (Section 2.4.2) to map *i* to the current/last index in the sequence, allowing the planner to track the order of readings.

Next follows the section `(:functions...)` for numeric fluents (numeric variables) of the domain. It includes two index variables `(current_index)`, with its obvious semantics, and `(last_index)`, which the planner uses to infer that the sequence of readings is over and there are no further predicates to be processed (for the specific location). The last function is `(threshold ?s - numeric)`, which associates the threshold of a numeric sensor with a numeric value. The predicate `(new_threshold ?s...)` updates the value of `(threshold ?s)`, but `(threshold ?s)` is not associated with an index and is applicable to both binary and numeric sensors. Therefore, the idea is to use `(new_threshold ?s...)` only when `?s` needs to update its threshold.

5.3 Tasks for recognising activities

Now we come to the main part of the activity recognition, namely the compound tasks to achieve this.

5.3.1 Sensor states

We first introduce the compound task for updating the sensor states according to their current reading, as to whether they are *active* or not, *involved* in the environment or not. We need to consider binary and numeric sensors. A binary sensor reports either state 1 or 0, which can be directly mapped to active or not active. A numeric sensor is a bit more complex, but it can also be translated into active or not. If the reported value of a numeric sensor is equal to or greater than its threshold, it is active, and when it is below the threshold, it is not.

We can see the HPDL representation in Listing 5.3. A HTN compound task is mapped by using the `(:task...)` construct, and its decomposition methods by `(:method...)` - Listing 2.2. An inference `(:inline...)` task is used to appropriately change the `(sensor_active ?s)` predicate.

5.3.2 Processing the sequence of sensor readings

We now come to the root compound task, the compound task that is to be initially set as a task network (Definition 2.2.3) to be decomposed (Definition 2.3.2) - Listing 5.4.

Its parameters input are the locations, where activities are happening, in this example again the working and meeting room, and the coffee corner. It has one decomposition method with an empty precondition, and it adds a task to recognise activity for each ?*l* from the input parameters of `(:task recognise_activities...)`.

It is important to note how the subtasks of its method are ordered, namely

```
(:method recognise_activities_in_parallel :tasks[] ).
```

That is the *unordered* subtask sequence - Section 2.5. This removes any temporal constraints between the primitive actions, added by one subtask, and the primitive actions, added by another. The planner first processes the sequence of reading predicates for `?working_room`, adds its recognised activities with their temporal start and end points, and then proceeds to `(recognise_activities_in ?meeting_room)`. If the *ordered* sequence () is used, then their temporal points must come after the last action, added for `?working_room`. The unordered sequence allows them to begin at their own starting point, as the planner goes through the readings sequence again.

The task `(:task recognise_activities_in ?l)` consists of first handling the sensor information for *current_index* and then trying to infer whether new activities are taking place at ?*l*, as well as a base case task that is used when the sequence of predicates is over.

Listing 5.5 shows its methods, used to process the series of `(reading...)` for a given ?*l*. Its first method is for capturing new threshold values for numeric sensors at ?*l* at the *current_index ?i*. Then `(:method current_index_sensors...)` looks for any new predicates `(reading ?s ?value ?i)`,

Listing 5.3 Compound task for updating sensor states

```
(:task update_sensor
    :parameters(?s - sensor ?value - number)
    ; a binary sensor is either active 1 or inactive 0
    (:method binary
        :precondition(and (is_binary ?s) (= ?value 1))
        :tasks (
                (:inline () (sensor_active ?s))
        )
    )
    (:method binary_negate
        :precondition(and (is_binary ?s) (= ?value 0))
        :tasks (
                (:inline () (not (sensor_active ?s)))
        )
    )
    ; numeric sensor ?s has a value that has reached its threshold, i.e. is active 1
    (:method numeric_threshold
        :precondition(is_numeric ?s)
        :tasks(
                (:inline (or (> ?value (threshold ?s)) (= ?value (threshold ?s)))
                        (sensor_active ?s))
        )
    )
    ; numeric sensor ?s has a value below its threshold and is therefore inactive 0
    (:method numeric_below_threshold
        :precondition(is_numeric ?s)
        :tasks(
                (:inline (< ?value (threshold ?s)) (not (sensor_active ?s)))
        )
    )
)
```

Listing 5.4 Root compound task for the domain

```
(:task recognise_activities
        :precondition()
        :parameters(?working_room - working_room ?meeting_room - meeting_room
                    ?coffee_corner - coffee_corner)
        (:method recognise_activities_in_parallel
            :tasks[
                    (recognise_activities_in ?working_room)
                    (recognise_activities_in ?meeting_room)
                    (recognise_activities_in ?coffee_corner)
            ]
        )
)
```

where ?*s* is again at ?*l*. If the precondition of the method evaluates to true, then the planner triggers the `(:task update_sensor ?s ?value...)` from Listing 5.3 and after it is completed, it negates the `(reading ?s ?value ?i)` predicate - now it cannot be processed again and cause a loop in the search of the planner. The next `(:method recognise_activity...)` captures the temporal information of the sensor readings in its precondition and calls its own compound task `(recognise_activity ?l ?t)`, at which we are later look. After it is completed, the *current_index* is incremented by one, as to tell the planner to look at the readings at the next index.

The first three methods in Listing 5.5 have `(:task recognise_activities_in ?l)` as their last subtask, thus introducing recursion in the task decomposition. SIADEX calls the methods of a task in a sequential order, that is `(:method current_index_sensors)` is called only after the precondition of `(:method update_threshold)` fails, and so on. Once the threshold has been updated (if needed), then the sensor readings are processed, and then the dedicated compound task to recognise activities is called.

The last method of `(recognise_activity ?l)` is the base case that is reached after the sequence for ?*l* has been processed, therefore it has an empty precondition. The `(:inline...)` tasks captures in its precondition the last activity that is happening at ?*l*, and its start and end time points. Then that activity is added to the plan by the primitive action `(recognised_activity ?a ?l)` with its temporal points *?a_start* and *?a_end*. Finally, the *current_index* is reset to 0, as the sequence is now over and the planner has to start at index 0 for `(recognise_activity ?l')` for the next location ?*l′*. It is important to note that we have to be cautious with *?a_start* and *?a_end*, in particular they must not violate the temporal constraints of the already constituted plan. If, for example, *?a_start* is before the end point of the last already added activity for ?*l*, then the task decomposition would fail and the planner would backtrack, searching for a solution in an endless loop. This can be avoided by another method, which only resets the index. In reality, this violation does not make sense, as the points of consequent readings also have to progress in time. This loop can also only happen at this point.

5.3.3 Activity recognition

As already indicated, the compound task for the activity recognition itself is `(recognise_activity ?l ?t)`, where ?*l* is is the location and ?*t* the time mark of the current sensor readings. Here is the point where we can define the hierarchy of the activities themselves, group them, and refine them in different sequences of subtasks.

For any particular activity, we have to define two methods - one for the situation where this activity is already taking place, where for `(current_activity ?l ?a)`, ?*a* is already this particular activity, and one for when ?*a* is another activity. Both methods have the same precondition except for the first including also `(current_activity ?l ?a)`.

The reason for this is that we want the planner to continue searching at the next index and not add ?*a* to the plan, in case it is the current activity - ?*a* is already saved in `(current_activity ?l ?a)`. When the planner recognises another activity ?*a′*, it can add ?*a* to the plan with its end and beginning points - its beginning point is saved in the predicate `(start_activity ?l ?timepoint)`, and its end point is when ?*a′* was recognised, which is simultaneously the beginning of ?*a′*. This way the planner can infer the start and end points of each recognised activity at ?*l*.

Listing 5.5 Methods for handling the sensor information sequence

```
(:method update_threshold
    :precondition(and (index ?i) (new_threshold ?s ?value ?i) (sensor_at ?s ?l))
    :tasks(
        (:inline () (and (assign (threshold ?s) ?value) (not (new_threshold ?s ?value ?i))))
        (recognise_activities_in ?l)
    )
)
(:method current_index_sensors
    :precondition(and (index ?i) (sensor_at ?s ?l) (reading ?s ?value ?i))
    :tasks(
        (update_sensor ?s ?value)
        ; (sensor ?s ?l ?value ?i) has been processed, delete from state to avoid loops
        (:inline () (not (reading ?s ?value ?i)))
        (recognise_activities_in ?l)
    )
)
(:method recognise_activity
    :precondition(and (index ?i) (timepoint ?t ?i))
    :tasks(
        (recognise_activity ?l ?t)
        (:inline () (increase (current_index) 1))
        (recognise_activities_in ?l)
    )
)
(:method done
    :precondition()
        :tasks(
            (:inline (and (current_activity ?l ?a)
                          (begin_activity ?l ?a_begin)
                          (last_index ?last_index) (timepoint ?a_end ?last_index)) ())
            (and (= ?start ?a_start) (= ?end ?a_end)) (recognised_activity ?a ?l))
            (:inline () (assign (current_index) 0))
        )
)
```

We can see an example for that in Listing 5.6. The only difference in the precondition of `(:method absence_continuous...)` is the additional predicate `(current_activity ?l absence)`, which indicates that *absence* is already taking place at *l*. It has an empty sequence () of subtasks, since we want the planner to continue with the search in this case - `(recognise_activity ?l ?t)` is accomplished and *current_index* is increased. We can also see how a certain activity, in this case *absence*, is to be recognised - each activity ?*a* at ?*l* is mapped to a combination of states of sensors at ?*l*. *Absence* is intuitively defined when there are no active sensors at ?*l*.

By following this principle, we can also pose more complex relationships, as grouping activities by their hierarchical dependencies, for example the meeting room activities from [NRA13], shown in Listing 5.7. They represent the simple hierarchy of *presence* in the *meeting room*. It can either be

Listing 5.6 Presence recognition methods

```
(:method absence_continuous
    :precondition(and(current_activity ?l absence)
                     (forall (?s - sensor) (imply (sensor_at ?s ?l) (not (sensor_active ?s))))
                 )
        :tasks()
)
(:method absence
    :precondition(forall (?s - sensor) (imply (sensor_at ?s ?l) (not (sensor_active ?s))))
        :tasks(
            (add_activity ?l absence ?timepoint)
        )
)
```

presence (users are present in the meeting room, but no specific activity can be determined), or *meeting*, where this is further refined in either *meeting* or the more specific kind of meeting *giving presentation*.

First, `(:method presence_meeting_room)` is activated when ?*l* is of type *meeting_room*. The compound `(:task recognise_presence_meeting_room)` then attempts to decompose the *meeting* activities by capturing the involved sensors for *meeting* in its precondition and passing them to the corresponding compound `(:task recognise_meeting...)`. If this decomposition fails, then the planner tries to identify the *presence* activity by `(:method presence_continuous...)` and `(:method presence...)`.

As all the relevant sensors are passed as parameters to `(:task recognise_meeting...)`, we can use directly the predicate `(sensor_active ?s)` in the precondition of its methods. When a single compound task has methods that identify multiple single activities, we can handle the case of when they are already happening in a single method, instead of defining it for each activity separately, as in the precondition of `(:method meeting_continuous...)`.

All the activities are finally added by `(add_activity ?l ?current_activity ?timepoint)`, where *?timepoint* is simultaneously the beginning of *?current_activity* and the end of the previous activity ?*a*. The following inference tasks help `(add_activity)` to add the primitive activity action with its temporal information:

```
(:inline (and (current_activity ?l ?a) (start_activity ?l ?a_start)) ())
((and (= ?start ?a_start) (= ?end ?timepoint)) (recognised_activity ?l ?a))
(:inline () (and (not (current_activity ?l ?a)) (not (start_activity ?l ?a_start))))
(:inline () (and (current_activity ?l ?current_activity) (start_activity ?l ?timepoint)))
```

We follow the same process, described for `(:method done...)` from Listing 5.5. At the point of the first subtask, the previous activity is saved in `(current_activity ?l ?a)`, the second subtask adds the previous activity to the plan, and the following tasks update the activity predicates.

Listing 5.7 Activities at meeting room recognition methods

```
(:method presence_meeting_room
    :precondition(is_meeting_room ?l)
    :tasks(
        (recognise_presence_meeting_room ?l - meeting_room ?timepoint)
    )
)
(:task recognise_presence_meeting_room
    :parameters(?l - meeting_room ?timepoint - number)
    (:method meeting
        :precondition(and (sensor_at ?chair1 - chair ?l) (sensor_at ?chair2 - chair ?l)
                          (not (= ?chair1 ?chair2)) (sensor_at ?projector - projector ?l)
                          (sensor_at ?acoustic - acoustic ?l))
        :tasks(
            (recognise_meeting ?chair1 ?chair2 ?projector ?acoustic ?l ?timepoint)
        )
    )
    (:method pesence_continuous
        :precondition(and (current_activity ?l presence)
                          (sensor_at ?infra - infrared ?l) (sensor_active ?infra))
        :tasks()
    )
    (:method presence
        :precondition(and (sensor_at ?infra - infrared ?l) (sensor_active ?infra))
        :tasks (
            (add_activity ?l presence ?timepoint)
        )
    )
)
(:task recognise_meeting
    :parameters(?chair1 ?chair2 ?projector ?acoustic - sensor ?l - meeting_room
                ?timepoint - number)
    (:method meeting_continuous
        :precondition(or (and (current_activity ?l giving_presentation)
                              (sensor_active ?chair1) (sensor_active ?chair2)
                              (sensor_active ?projector) (sensor_active ?acoustic))
                         (and (current_activity ?l meeting)
                              (sensor_active ?chair1) (sensor_active ?chair2)
                              (not (sensor_active ?projector)) (sensor_active ?acoustic)))
        :tasks()
    )
    (:method giving_presentation
        :precondition(and (sensor_active ?chair1) (sensor_active ?chair2)
                          (sensor_active ?projector) (sensor_active ?acoustic))
        :tasks(
            (add_activity ?l giving_presentation ?timepoint)
        )
    )
    (:method having_meeting
        :precondition(and (sensor_active ?chair1) (sensor_active ?chair2)
                          (not (sensor_active ?projector)) (sensor_active ?acoustic))
        :tasks(
            (add_activity ?l meeting ?timepoint)
        )
    )
)
```

Listing 5.8 An example SIADEX solution plan for a problem

```
:action (recognised_activity working_room absence) start: 0 end: 80000
:action (recognised_activity working_room presence) start: 80000 end: 81000
:action (recognised_activity working_room working_wout_pc) start: 81000 end: 82000
:action (recognised_activity working_room working_with_pc) start: 82000 end: 100000
:action (recognised_activity working_room absence) start: 82500 end: 100000
:action (recognised_activity meeting_room absence) start: 0 end: 93000
:action (recognised_activity meeting_room meeting) start: 93000 end: 100000
:action (recognised_activity coffee_corner absence) start: 0 end: 100000
:action (recognised_activity coffee_corner presence) start: 95000 end: 100000
```

The only primitive action in our domain, as the actions that make up a solution to an instance of a planning problem (Definition 2.3.3), is `(:action recognised_activity :parameters(?l - location ?a - activity) :effect())`.

In the end, the plan of recognised activities for a $?l$ is a temporally ordered sequence of `(:action recognised_activity)`, where the different recognised activities are distinguished by the parameter $?a$. Overall, the solution of a planning problem instance is again a sequence of `(:action recognised_activity)`, where first is the sequence of actions for l_1, then for l_2, etc. As described in the beginning of Section 5.3.2, for each a and a' at l_i, where a is added before a', it follows that $\text{end}(a) \leq \text{start}(a')$. For a at l_i and a' at l_j, there are no temporal constraints between a and a'.

5.4 Problem instance of the domain

The problem defines the initial state of the planning process, as in Listing 2.1. We define the used sensors and locations in the `(:objects)` section, and then proceed to `(:init)`. There we need to use the `(sensor_at ?s ?l)` predicate to specify the location of each sensor *?s*, set the threshold of *?s* if it numeric, and provide `(current_activitiy ?l ?a)` for the initial activity at each location *?l*.

The main part of the problem consist of the `(reading ?s ?value ?i)` and the corresonding `(timepoint ?t ?i)` predicates, which capture the sequence of sensor events. We set `current_index` to 0 and `last_index` to k, with k being the last index in the sequence.

The initial task to be decomposed is `(recognise_activities` $l_1, \ldots, l_n$`)` for n different locations. The solution to a problem, the decomposition of the initial task, is a plan of temporally annotated recognised activities for the different l_i. We can see an example for the output of SIADEX in Listing 5.8. We have to use plain numeric fluents for the representation of time, as our version of SIADEX [Ign] does not support the timed initial literals from [FP20].

6 Validation & Evaluation of Solution

To evaluate the capabilities of HPAR, we test the proposed framework with input, gathered from real experiments, where simple sensors were deployed in a real home environment.

6.1 Experimental setup

A public dataset that fulfils our requirements is provided for the experiments in van Kasteren et al. [KEK11]. The dataset consists of several weeks of data recorded in a real world setting. Wireless networks of simple non-intrusive sensors were installed in the homes of three different experiment subjects, hence three different experiments were conducted and three different datasets of sensor observations were gathered. All sensors used give binary output.

The download link for the datasets in [KEK11] is not functional anymore, but luckily, a repository containing the experiment data from [KEK11] is provided in [AA18]. However, the files in the repository are in a different format (*.csv*), and for the second and third experiment, the sensor readings are already annotated with the corresponding events. To the best of our knowledge, the original files for [KEK11] can only be found for the first experiment, therefore we choose this dataset for our evaluation.

6.1.1 Kasteren House A

The experiment for house *A* includes 1319 sensor readings from 14 distinct sensors, spread across three rooms. The data was recorded in 25 different days in the apartment of a 26 years old male. The subject annotated his current activity by the means of speech recognition of predefined commands to a bluetooth headset.

Eight distinct activities are to be recognised - '*leave house*', when the subject leaves or enters the house, '*use toilet*' and '*take shower*', *'go to bed'*, when the subject enters or leaves his bedroom, and the activities, performed in the kitchen - '*prepare breakfast*', '*prepare dinner*', and '*get drink*'. A last activity is implicitly '*absence*', when the subject is currently not performing any activity, this corresponds to '*None*' from the annotated files in [AA18].

Interestingly, this dataset presents a possibility to evaluate activities with a different granularity of information about them in the dataset. For example, only the '*Hall-Bedroom door*' is tagged with a sensor and consequently the only information we can make assumptions about whether the user is currently performing '*go to bed*' is the activation of the '*Hall-Bedroom door*' sensor. On the other hand, various appliances - '*microwave*', '*fridge*', '*dishwasher*', '*cups cupboard*', . . ., are equipped

with a sensor in the kitchen, making the information about the kitchen activities '*prepare breakfast*', '*prepare dinner*', '*get drink*' much more richer. However, the recognition of these activities involves very similar artifacts and makes their interpretation ambiguous.

6.1.2 Our Model

We implement a *House A* HPDL domain, based on the model, discussed in Chapter 5. We have 5 compound tasks `(recognise_activities_in ?l)` as in Listing 5.4, namely for ?*l* as *?house*, *?bedroom*, *?toilet*, *?bathroom*, and *?kitchen*. For the first four locations there is only one specific corresponding activity, namely *?house 'leave house'*, *?bedroom 'go to bed'*, *?toilet 'take shower'*, and *?bathroom 'use toilet'* (the mapping of the latter two might seem a bit confusing, but that is the way the sensors were named in the experiment). In *?kitchen* we have *'prepare breakfast'*, *'prepare dinner'* and *'get drink'*.

As each of the sensors reports a binary value, we do not need to consider numeric sensors and their thresholds. Furthermore, we build the problem instance for the domain only on the basis of activation readings of the sensors, that is when the sensor switches from 0 to 1. We exclude predicates for when the sensor becomes inactive, as from 1 to 0, as it is usually right after the activation, but other times the sensor remains active for an unreasonable amount of time. For example, let us consider the following sensor reading:

Start time: *25-02-2008 09:33:47* **End Time**: *25-02-2008 17:21:12* **Hall-Bedroom door 1**

It indicates that *'Hall-Bedroom door'* is activated at *09:33:47* and remains active until *17:21:12*, which introduces a lot of confusion if we try to interpret the time when it becomes inactive.

The different states of a sensor and the mapping to a specific activity based on which sensors are active/inactive might be very useful for domains like in [NRA13], where a transition of sensor states is also associated with a new activity. For example, when the user switches of his computer and monitor, but is still sitting on his chair, we transition from the activity '*Working with PC*' to '*Working without PC*'.

Our model is designed to cover both binary and numeric sensors. In the case of *House A*, we are only interested in the current active sensor, and we adapt our model to only consider this information when the planner updates the predicates for the sensors. Other than that, the model remains the same.

6.1.3 Comparison of results

As mentioned, the dataset for *House A* consists of 1319 sensor readings. A labeled dataset, where the timestamp of the sensor reading corresponds to the activation time of the sensor, is provided for evaluation and comparison of results. Each reading is labeled with the activity that was associated with the reading, as for example `2008-02-25 10:02:31 HallToiletDoor ON TakeShower`. The activity label is inferred by the annotations provided by the subjects. In case no activity was taking place at the time of the sensor reading, which implies that this was an incorrect sensor activation, the reading is labeled with '*none*'.

Since the result of our planning problem is a plan with a list of recognised activities at each location, we implement a small script to label the original log of sensor readings according to the activities that the planner has recognised. For example, let us consider a reading r of a sensor s at timestamp t. We are now interested in an activity a at l, so that s is at l, $start(a) >= t$ and $end(a) < t$, and label r with a. If no such activity exists, then we label r with *absence*. For comparison, if r is originally labeled with *None*, we expect to label it with *absence*.

Note that the domain model can easily be adapted to label each reading that is processed, that is the planner adds a primitive action with the current activity for every sensor reading.

6.1.4 Evaluation metrics

We compare our labeled sensor events log with the provided ground truth. Based on the total number of instances of an activity in the ground truth, and the number of corresponding true and false positives, we calculate the *precision, recall* and *F-score* of every recognised activity. Furthermore, we calculate these metrics also for the *None* activity, or the *asbence* activity, that is when the sensor activation was not related to any activity.

We calculate the mentioned metrics for every of the 25 observed days in the data log. To ease the evaluation of the achieved results, we assign the following values, should one of the metrics not be applicable by definition. If for a given day there are no instances of an activity in the ground truth, as for example the subject did not make dinner because he was out of the house, we look at the labels we have assigned for this day - if our results also did not label any sensor reading with this activity, we assign 1 to the precision, recall and F-score for this activity for this day, else 0. In the other direction, if we labeled readings with a given activity, but it was not found in the ground truth, we assign 0 to each metric.

6.2 Results

6.2.1 Execution

Unfortunately, to the best of our knowledge, our version of SIADEX, installed from the public repository of the authors [Ign], does not support PDDL timed initial predicates, or facts embodying time data, as they were used in [FP20]. Our version cannot parse strings like `"25/02/2008 09:33:47"`.

Therefore, we split the readings from the log day by day, and create a separate problem instance with the relevant readings for each of the 25 observed days. For the representation of time literals, we use plain numeric fluents, as `93347` for `09:33:47`. Note that if we did not split the readings by day, the planner would not be able to constitute a temporal plan. The end of the last activity of the previous day is later than the start of the first activity of the following day, and we lack date stamps to indicate that this is not a temporal violation.

We run the planner with a problem instance for each of the 25 days, and evaluate the observed recognised actions.

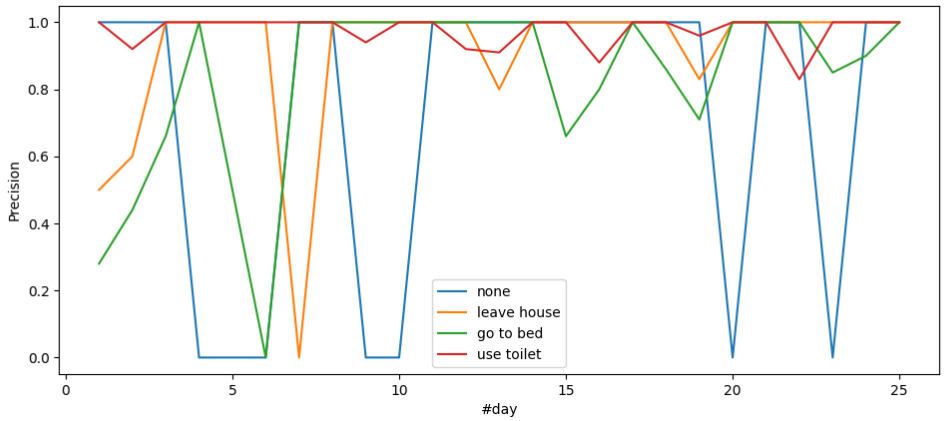

Figure 6.1: Precision for 'none', 'leave house', 'go to bed' and 'use toilet'

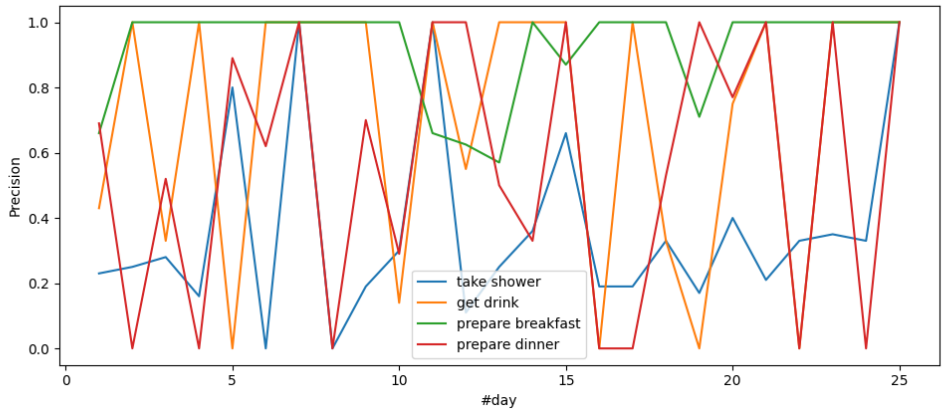

Figure 6.2: Precision for 'take shower', 'get drink', 'prepare breakfast' and 'prepare dinner'

6.2.2 Precision, Recall and F-score

In Figure 6.1 and Figure 6.2 we can see the achieved percentages of precision throughout the experiment days for the eight distinct activities. Figure 6.3 and Figure 6.4 show us the results for recall. The last day of the experiment includes only one reading, which is recognised correctly. Therefore, all of the activities end at score 1. Figure 6.5 represents the total average mean of the achieved percentages of the metrics, including the F-score.

6.2.3 Discussion of the results

It is evident from the figures for precision and recall that our method does not use any deep learning or reasoning about the input data, or the sequences of sensor readings. The percentages for every activity go up and down, not following a certain pattern.

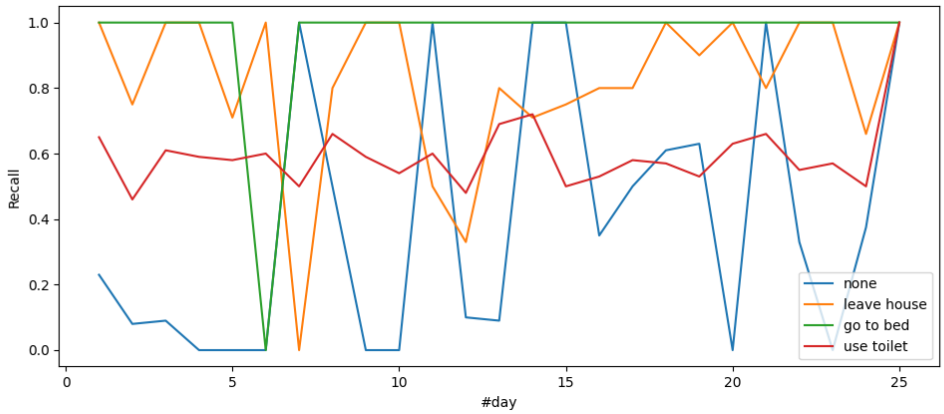

Figure 6.3: Recall for 'none', 'leave house', 'go to bed' and 'use toilet'

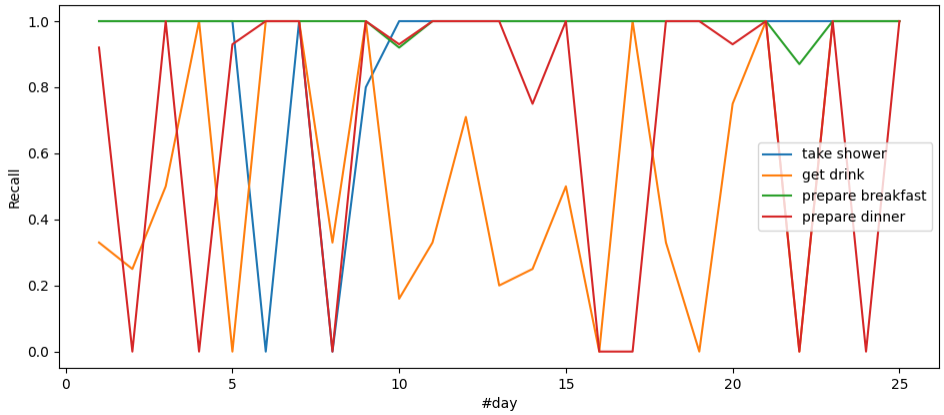

Figure 6.4: Recall for 'take shower', 'get drink', 'prepare breakfast' and 'prepare dinner'

For example, the precision for '*none*' (Figure 6.1) goes from 1 to 0 and from 0 to 1, not holding any other value in between. That is, we either recognise correctly all instances of sensor readings that did not belong to any activity, or not even a single one. The curves for '*leave house*', '*go to bed*' also make drastic changes, although not so extreme. A good reason for this is that there is only one sensor for each of these activities to infer whether the activity is taking place or not, as '*Hall-Bedroom door*' for '*go to sleep*'. Consequently, for each activation of '*Hall-Bedroom door*', we deduce that the subject is performing *'go to sleep'*. Irrelevant sensor activations are quite common in the dataset, for example this sequence of annotations from the ground truth:

```
2008-02-25 19:21:34 HallBedroomDoor ON None
2008-02-25 19:21:39 HallBedroomDoor ON None
```

We do not make any assumptions about the error probability of a sequence of readings. This implies that the precision and recall of labeled readings for such activities relies solely on how accurately the involved sensor publishes data.

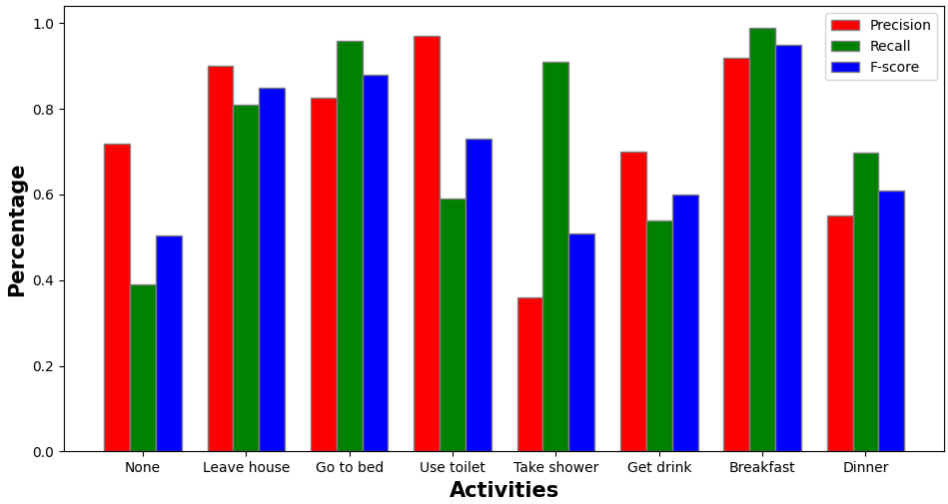

Figure 6.5: Average values for precision, recall and F-score for the different activities

We can observe good precision scores for '*use toilet*' and '*prepare breakfast*', and constantly changing curves for '*take shower*', '*get drink*' and '*prepare dinner*'. Big jumps in the curves are present for every activity except '*use toilet*', often achieving good recognition results on one day, and performing bad on the following, and vice versa.

Going to the recall, the amplitudes between the different points of the curve are even more evident. We achieved very good recall results for '*go to bed*' and '*prepare breakfast*', as the planner was able to recognise almost every instance of both activities. Here the scores for '*none*' are a bit more variable, but therefore the scores for more activities make switches between 1 and 0 for recall. A big reason for them is also the convention we make to assign 0 or 1 to all metrics, when they are not applicable by definition. The obtained scores, however, are representative of the achieved results of recognition.

Overall, Figure 6.5 is representative of the achieved recognition performance. A very impressive F-score of 0.95 was achieved for '*prepare breakfast*', followed by '*go to bed*' with 0.88 and '*leave house*' with 0.85. On the lower end, we have '*prepare dinner*' and '*get drink*' with 0.61 and 0.6, respectively, and '*take shower*' and '*none*' with 0.51 and 0.5. Although the precision of 0.97 for '*use toilet*' was very high, the percentage of overall recognised instances was only 59%, resulting in a F-score of 0.73.

Keeping in mind that the used model is simple and does not incorporate any advanced statistical methods, the achieved results are satisfying. Due to the temporal capabilities of SIADEX, the planner was able to distinguish between similar activities like '*prepare breakfast*' and '*prepare dinner*'. Since they are performed during different times of the day, the planner can easily check if the current timepoint is more relevant for breakfast or for dinner. However, we can see that it becomes much more difficult when also the time spans of similar activities overlap, as with '*get drink*' and '*prepare dinner*'. Both have a similar F-score, the results for '*get drink*' were more precise, whereas more instances of '*prepare dinner*' were recognised.

Figure 6.6: Comparison of the performance of different approaches for House A [AA18]

Category	Approach	Precision	Recall	F-score
Hybrid	ARF	88.6%	95.48%	91.91%
Hybrid	HARS	75.97%	78.57%	77.01%
Hybrid	USMART	*	*	74.0%
Knowledge	HPAR	74.32%	73.6%	73.95%
Supervised	HSMM	70.5%	75.0%	72.4%
Supervised	HMM	70.3%	74.3%	72.0%
Supervised	NB	67.3%	64.8%	65.8%
Supervised	CRF	73.5%	68.0%	70.4%

6.2.4 Comparison with other results for Kasteren House A

There are a number of other experiments with the dataset of Kasteren House A. We can see an informative overview of achieved results of the different approaches in Figure 6.6 [AA18]. The results of the supervised methods are these of the original authors of [KEK11]. They use Naive Bayes (*NB*), Hidden Markov Model (*HMM*), Hidden semi-Markov Model (*HSMM*) and Conditional Random Field (*CRF*). Given that our model is entirely knowledge-based, and our achieved percentages of 74.32% for precision, 73.6% for recall, and the average F-score of 73.95%, it performs notably well next to much more complicated supervised learning techniques.

Going back to the unsupervised learning model of USMART [YSD14], the authors provide only the average F-Score of 74.0% for the House A dataset, but it has to be noted that they do not consider the '*none*' activity, since they need at least one sensor event that uniquely describes an activity, which is obviously not present in the case of '*none*'. Therefore the obtained results from USMART are not entirely comparable to ours.

USMART is outperformed by HARS [AA18]. Although lightweight, the framework is able to label sensor events with a precision of 75.97% and recall of 78.57% on average, and an F-Score of 77.01%. The only better performing system is that of the Activity Recognition Framework of [INIT18]. Its results are considerably higher than any other reported result with a precision score of 88.6%, recall of 95.48%, and an impressive F-score of 91.91%. However, we have to keep in mind that this framework requires serious modelling efforts, which might reduce its effectiveness in a real world scenario.

6.3 Computational load

The planning time of SIADEX never lasted more than a few seconds, although the planner run in a virtual machine that was assigned only one GB of memory and one CPU core. The domain consists of eight distinct activities happening at five different locations, which is a fair representation of a real world scenario. We had to split the sensor events log in days, because of the temporal representation limitations, but the amount of predicates added to the planning problem file did not influence significantly (as more than linearly) the execution time of the planner.

Our domain does not consider the specific users who are performing the activities, and further tests are needed to evaluate properly its computational performance.

7 Conclusion and Outlook

In conclusion, we discuss the most important findings of our work.

7.1 Hierarchical planning for activity recognition

In Chapter 4, we explored how we can translate sensor events from real world data or already extracted datasets into a generic hierarchical planning language. The main features to be modeled are thoroughly discussed, and ways to integrate activity recognition methods in the domain knowledge are presented. Chapter 5 describes how these ideas are implemented in a concrete hierarchical domain model, in our language of choice HPDL [Geo13]. The result is an actual temporal HTN problem, aimed at recognising activities.

We show that temporal HTN planning can be practically executed by deploying the publicly available hierarchical planner SIADEX [FCGP06]. We apply it in Chapter 6 to a dataset, extracted from a real world experiment [KEK11].

Although below the achieved scores for any of the hybrid approaches, our framework performed better than the initial supervised learning methods of [KEK11]. This shows that the suggestion of Philipose et al. [PFP+04] that the object a subject interacts with is indeed a major indicator for the current activity of the subject, as we are able to obtain comparable results to supervised methods.

Moreover, our results are not significantly worse than the reported results of USMART [YSD14] and HARS [INIT18]. Actually, in comparison to USMART, we are even able to distinguish false sensor events that do not belong to any activity, as is the case with '*none*'.

Another thing to point out is that our model is not designed with any specific dataset in mind. Its purpose is to recognise activities and constitute a plan of activities with temporal timepoints. It is more of a general idea of how to represent activity recognition in the setting of an HTN planning problem, rather than a sophisticated activity recognition solution. Let us consider the following sequence of 32 sensor events from the ground truth of the House A Kasteren experiment:

```
2008-03-06 19:53:38 Dishwasher ON None
2008-03-06 19:53:44 Dishwasher ON None
2008-03-06 19:54:35 PlatesCupboard ON None
2008-03-06 19:54:46 PansCupboard ON None
...
2008-03-06 21:37:18 GroceriesCupboard ON None
```

Each of the 32 events is labeled with '*none*', which means that there were 32 consequent false sensor activations in the kitchen, as the user did not actually interact with them.

Our model and HTN planning in general is not meant to make probabilistic estimations about the input data. To our domain, this sequence means that the user prepared dinner from the start until the end. However, the capability to identify when such sequences are likely to not belong to any activity at all would be of great usefulness in such situations, as incorrect sensor readings are bound to happen.

We see great potential in activity recognition by hierarchical planning, especially if we further elaborate this concept and delegate the parsing of the sensor events log to another process, which employs learning algorithm. This would also make the planning problem closer to the paradigm of classical planning, as we do not have to consider fine details like following a specific sequence of predicates, thus forcing the planner to follow a certain search pattern.

We also hope that the comparison to existing approaches in Chapter 3, and especially how hierarchical planning can extend or ease the modelling of activities on the modelling part of the system, show the promising qualities of HTN planning.

Outlook

As of now, HPAR can only be executed offline, as we include all of the sensor events in the problem instance. That is, the data has to be first collected, and then translated to an HPDL problem.

An interesting perspective is to adapt the model design to consider sensor events from an ongoing data stream. This frees the planner from reading over the complete sensor log, and makes the domain closer to the paradigm of classical planning. In this case, the planner has to consider predicates that only represent the current state of the observed environment, without having to adapt the state between different sensor readings.

An even more intriguing possibility, as discussed in Section 7.1, is to integrate hierarchical planning into a hybrid approach, where data-driven techniques are used to segment and synthesize the sensor readings, and activity recognition on a higher level is executed as a HTN planning problem. This can improve by a significant margin the obtained results in this work, and motivate further research on this topic.

Bibliography

[AA18] G. Azkune, A. Almeida. "A Scalable Hybrid Activity Recognition Approach for Intelligent Environments". In: *IEEE Access* 6 (2018), pp. 41745–41759. DOI: 10.1109/access.2018.2861004 (cit. on pp. 14, 35, 53, 59).

[AHK+98] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. SRI, A. Barrett, D. Christianson, et al. "PDDL| The Planning Domain Definition Language". In: *Technical Report, Tech. Rep.* (1998) (cit. on p. 23).

[BG00] B. Bonet, H. Geffner. "Planning with Incomplete Information as Heuristic Search in Belief Space." In: Jan. 2000, pp. 52–61 (cit. on p. 17).

[BHP+06] J. Boger, J. Hoey, P. Poupart, C. Boutilier, G. Fernie, A. Mihailidis. "A Planning System Based on Markov Decision Processes to Guide People With Dementia Through Activities of Daily Living". In: *IEEE Transactions on Information Technology in Biomedicine* 10.2 (Apr. 2006), pp. 323–333. DOI: 10.1109/titb.2006.864480 (cit. on pp. 40, 41).

[BHS+20] G. Behnke, D. Höller, A. Schmid, P. Bercher, S. Biundo. "On Succinct Groundings of HTN Planning Problems". In: *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020)*. AAAI Press, 2020 (cit. on pp. 24, 43).

[CFGP06] L. A. Castillo, J. Fernández-Olivares, Ó. García-Pérez, F. Palao. "Efficiently Handling Temporal Knowledge in an HTN Planner." In: *ICAPS*. 2006, pp. 63–72 (cit. on pp. 25–27).

[CHN+12] L. Chen, J. Hoey, C. D. Nugent, D. J. Cook, Z. Yu. "Sensor-Based Activity Recognition". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (Nov. 2012), pp. 790–808. DOI: 10.1109/tsmcc.2012.2198883 (cit. on p. 30).

[CK11] L. Chen, I. Khalil. "Activity Recognition: Approaches, Practices and Trends". In: *Activity Recognition in Pervasive Intelligent Environments*. Atlantis Press, 2011, pp. 1–31. DOI: 10.2991/978-94-91216-05-3_1 (cit. on p. 33).

[Ecl] Eclipse Foundation. *Eclipse Mosquitto™ - An open source MQTT broker*. URL: https://mosquitto.org (cit. on p. 38).

[FCGP06] J. Fdez-Olivares, L. Castillo, Ó. García-Pérez, F. Palao. "Bringing Users and Planning Technology Together. Experiences in SIADEX." In: Jan. 2006, pp. 11–20 (cit. on pp. 24, 61).

[FL03] M. Fox, D. Long. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research* 20 (Dec. 2003), pp. 61–124. DOI: 10.1613/jair.1129 (cit. on pp. 23, 24).

[FN71] R. E. Fikes, N. J. Nilsson. "Strips: A new approach to the application of theorem proving to problem solving". In: *Artificial Intelligence* 2.3-4 (Dec. 1971), pp. 189–208. DOI: 10.1016/0004-3702(71)90010-5 (cit. on p. 18).

[FP20] J. Fernandez-Olivares, R. Perez. "Driver Activity Recognition by Means of Temporal HTN Planning". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 30 (June 2020), pp. 375–383. DOI: 10.1609/icaps.v30i1.6683 (cit. on pp. 24, 26, 32, 33, 39, 52, 55).

[GA15] I. Georgievski, M. Aiello. "HTN planning: Overview, comparison, and beyond". In: *Artificial Intelligence* 222 (May 2015), pp. 124–156. DOI: 10.1016/j.artint.2015.02.002 (cit. on p. 19).

[Geo13] I. Georgievski. "HPDL: Hierarchical Planning Definition Language". In: *Uni. of Groningen, JBI Preprint* (2013), pp. 12–3 (cit. on pp. 24, 61).

[Geo15] I. Georgievski. "Coordinating services embedded everywhere via hierarchical planning". English. PhD thesis. University of Groningen, 2015. ISBN: 978-90-367-8148-0 (cit. on pp. 14, 17, 18, 20, 21).

[GNA13] I. Georgievski, T. A. Nguyen, M. Aiello. "Combining Activity Recognition and AI Planning for Energy-Saving Offices". In: *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*. IEEE, Dec. 2013. DOI: 10.1109/uic-atc.2013.106 (cit. on p. 34).

[GNT04] M. Ghallab, D. Nau, P. Traverso. *Automated Planning: Theory and Practice*. The Morgan Kaufmann Series in Artificial Intelligence. Amsterdam: Morgan Kaufmann, 2004. ISBN: 978-1-55860-856-6. URL: http://www.sciencedirect.com/science/book/9781558608566 (cit. on pp. 18, 19).

[HBB+20] D. Höller, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, R. Alford. "HDDL: An extension to PDDL for expressing hierarchical planning problems". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 06. 2020, pp. 9883–9891 (cit. on pp. 24, 25).

[Ign] Ignacio Vellido. *SIADEX - An HTN planner with temporal, partial order planning*. URL: https://github.com/IgnacioVellido/HPDL-Planner (cit. on pp. 52, 55).

[INIT18] I. Ihianle, U. Naeem, S. Islam, A.-R. Tawil. "A Hybrid Approach to Recognising Activities of Daily Living from Object Use in the Home Environment". In: *Informatics* 5.1 (Jan. 2018), p. 6. DOI: 10.3390/informatics5010006 (cit. on pp. 29, 35, 59, 61).

[KEK11] T. L. M. van Kasteren, G. Englebienne, B. J. A. Kröse. "Human Activity Recognition from Wireless Sensor Network Data: Benchmark and Software". In: *Activity Recognition in Pervasive Intelligent Environments*. Atlantis Press, 2011, pp. 165–186. DOI: 10.2991/978-94-91216-05-3_8 (cit. on pp. 53, 59, 61).

[LOB11] J. O. Laguna, A. G. Olaya, D. Borrajo. "A Dynamic Sliding Window Approach for Activity Recognition". In: *User Modeling, Adaption and Personalization*. Springer Berlin Heidelberg, 2011, pp. 219–230. DOI: 10.1007/978-3-642-22362-4_19 (cit. on p. 38).

[McD00] D. M. McDermott. “The 1998 AI Planning Systems Competition”. In: *AI Magazine* 21.2 (June 2000), p. 35. DOI: 10.1609/aimag.v21i2.1506. URL: https://ojs.aaai.org/index.php/aimagazine/article/view/1506 (cit. on p. 25).

[MQT] MQTT Organisation. *The Standard for IoT Messaging*. URL: https://mqtt.org (cit. on p. 38).

[NA13] T. A. Nguyen, M. Aiello. “Energy intelligent buildings based on user activity: A survey”. In: *Energy and Buildings* 56 (Jan. 2013), pp. 244–257. DOI: 10.1016/j.enbuild.2012.09.005 (cit. on p. 34).

[NBW] U. Naeem, J. Bigham, J. Wang. “Recognising Activities of Daily Life Using Hierarchical Plans”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 175–189. DOI: 10.1007/978-3-540-75696-5_11 (cit. on pp. 31–33, 35).

[NRA13] T. A. Nguyen, A. Raspitzu, M. Aiello. “Ontology-based office activity recognition with applications for energy savings”. In: *Journal of Ambient Intelligence and Humanized Computing* 5.5 (Sept. 2013), pp. 667–681. DOI: 10.1007/s12652-013-0206-7 (cit. on pp. 14, 33, 34, 37–40, 44, 49, 54).

[PFP+04] M. Philipose, K. Fishkin, M. Perkowitz, D. Patterson, D. Fox, H. Kautz, D. Hahnel. “Inferring Activities from Interactions with Objects”. In: *IEEE Pervasive Computing* 3.4 (Oct. 2004), pp. 50–57. DOI: 10.1109/mprv.2004.7 (cit. on pp. 13, 14, 29, 31, 38, 39, 61).

[VMW] VMWare, Inc. *RabbitMQ*. URL: https://www.rabbitmq.com (cit. on p. 38).

[Wei99] M. Weiser. “The computer for the 21st century”. In: *ACM SIGMOBILE Mobile Computing and Communications Review* 3.3 (July 1999), pp. 3–11. DOI: 10.1145/329124.329126 (cit. on p. 13).

[YSD14] J. Ye, G. Stevenson, S. Dobson. “USMART”. In: *ACM Transactions on Interactive Intelligent Systems* 4.4 (Nov. 2014), pp. 1–27. DOI: 10.1145/2662870 (cit. on pp. 35, 59, 61).

All links were last followed on January 30, 2023.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature