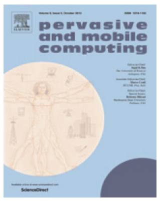
Pervasive and Mobile Computing 38 (2017) 110–139

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc



Planning meets activity recognition: Service coordination for intelligent buildings

Ilche Georgievski [*], Tuan Anh Nguyen, Faris Nizamic, Brian Setz, Alexander Lazovik, Marco Aiello

Johann Bernoulli Institute, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands

ARTICLE INFO

Article history:
Received 11 March 2016
Received in revised form 2 December 2016
Accepted 16 February 2017
Available online 20 February 2017

Keywords:
Service composition
AI planning
Activity recognition
Building automation

ABSTRACT

Building managers need effective tools to improve occupants' experiences considering constraints of energy efficiency. Current building management systems are limited to coordinating device services in simple and prefixed situations. Think of an office with lights offering services, such as turn on a light, which are invoked by the system to automatically control the lights. In spite of the evident potential for energy saving, the office occupants often end up in the dark, they have too much light when working with computers, or unnecessary lights are turned on. The office is thus not aware of the occupants' presence nor anticipates their activities. Our proposal is to coordinate services while anticipating occupant activities with sufficient accuracy. Finding and composing services that will support occupant activities is however a complex problem. The high number of services, the continuous transformation of buildings, and the various building standards imply a search through a vast number of possible contextual situations every time occupants perform activities. Our solution to this building coordination problem is based on Hierarchical Task Network (HTN) planning in combination with activity recognition. While HTN planning provides powerful means for composing services automatically, activity recognition is needed to identify occupant activities as soon as they occur. The output of this combination is a sequence of services that needs to be executed under the uncertainty of building environments. Our solution supports continuous context changes and service failures by using an advanced orchestration strategy. We design, implement and deploy a system in two cases, namely offices and a restaurant, in our own office building at the University of Groningen. We show energy savings in the order of 80% when compared to manual control in both cases, and 60% when compared to using only movement sensors. Moreover, we show that one can save a figure of €600 annually for the electricity costs of the restaurant. We use a survey to evaluate the experience of restaurant occupants. The majority of them are satisfied with the solution and find it useful. Finally, the technical evaluation provides insights into the efficiency of our system.

1. Introduction

The effective and efficient operation of buildings depends on the capabilities of their management systems. These are control systems for individual buildings (or groups of buildings) that use computers and other devices for monitoring, data

[*] Corresponding author.
E-mail address: i.georgievski@rug.nl (I. Georgievski).

http://dx.doi.org/10.1016/j.pmcj.2017.02.008
1574-1192/© 2017 Elsevier B.V. All rights reserved.

storage and communication [1]. In modern buildings, examples of devices include actuators, such as switches on ceiling lamps, movement detectors, and light sensors. Commonly considered capabilities of the building management systems include heating, ventilation and air condition control, lighting control, hot water control, and electricity control. The general objective of these systems is to improve occupants' experiences often under the constraints of energy efficiency [2].

With their current capabilities, building management systems can easily deal with triggering a single *service* or even a predefined sequence of services. Services here represent the functionalities of devices given that the development of building management systems is migrating to service-oriented architectures, as illustrated in [3,4]. An example of a service is turning on lights in modern office buildings using movement detectors. This building operation is supposed to reduce the energy consumption of lighting by 50% [5]. In these scenarios, the lights usually stay turned on until after some predefined time after detectors stop sensing movements. This operation is perfectly fine when the occupant, let us say Theodore, has left his office. Often, unfortunately, the lights go off while Theodore is still in the office and performs an activity, such as reading, for which the detector cannot observe his movements. The indication is that the building is neither aware of Theodore's presence nor anticipates his behaviour. The implication might be that the building does not even save as much energy as possible under such circumstances. For example, the activity of Theodore may require no light at all, or if it requires, the natural light might be sufficient. In any case, the building misses to provide intelligent responses.

The problem

Creating an aware and responsive building environment that enables occupants to have a truly immersive experience requires that the building management systems react to occupants with plausible levels of understanding of the situations and occupant activities. The systems can accomplish this by *service coordination*, which represents the task of composing a sequence of services while anticipating the activities of occupants. Service composition is the process of finding and arranging services in an order that satisfies a given objective. Service coordination motivates the following aspects to be considered. First, the approach to service coordination has to be able to deal with a high number of services. Say that a modern office building makes use of ten services for monitoring and control per desk. If the building has five floors each with 50 offices with two desks, as many as 5000 services can be expected, excluding those for corridors, social corners, meeting rooms, etc. Second, the approach needs to support the transformation of buildings. That is, devices may change, their services may constantly appear and disappear, and the state of the devices may continuously change. Third, the approach has to be able to incorporate and consider various requirements, from occupant needs to building standards. In office buildings, for example, many activities and therefore the performance of occupants depend on the quality of light in offices. For European countries, for instance, the basic light requirements are defined in the European standard for lighting in indoor work places [6]. Finally, the approach should consider as early and as many as possible activities of building occupants. Occupant behaviour has been shown to have a large impact on the demand of space heating, cooling and ventilation, lighting and appliances in buildings [7].

Considering these aspects, a *building coordination problem* involves a search through a high number of possible contextual states every time occupants perform some activities. Finding the compositions of services that will satisfy the requirements needed for performing the activities then becomes a complex task. A solution to this building coordination problem is to perform the service composition automatically and dynamically, assuming that occupant activities are identified as early as possible during the coordination. Doing this should maximise occupant comfort (or, similarly, minimise occupant disruption) possibly under the constraints of minimal energy consumption.

Once a sequence of services that anticipates the activities of occupants is found, the next step is to execute the services in the building environment. Therefore, the second problem that we address is how to efficiently execute services when the environment changes due to external factors, or services happen to fail at execution time.

Proposed solution

Existing control approaches show limited capabilities in addressing the complexity of building environments. First, preprogrammable timers and set points simply cannot address the requirements of buildings (e.g., dynamism, service availability, uncertainty, scalability, and support for standards, protocols and policies). Second, the existing platforms for pervasive computing applications, such as Jini (www.river.apache.org), appear to be limited in terms of providing only basic (device) interoperation [8]. Third, rule-based approaches, which provide more complex control functionalities, are characterised by several drawbacks: (1) a lack of flexibility — all possible situations that may happen in a building environment need to be predicted and covered by (a large number of carefully designed) rules; (2) a lack of systematicity — as the building environment becomes more robust, more rules are added without any enforced systematic steps resulting in a cluttered building management system; (3) a limited service order — creation of ordered or partially ordered adaptations appears to be a challenge (for example, given an objective to close a window, a rule-based approach might create a coordination consisting of, first, pulling down the blinds, and then closing the window); (4) a lack of reusability — reuse, modifications and maintenance across different types of building environments becomes difficult. Two spaces may happen to have some common situations, but if those spaces serve different purposes, say a meeting room and a social corner, then most of their situations will differ.

Our solution to the building coordination problem is based on a novel application of Artificial Intelligence (AI) planning to compose services. AI planning provides powerful means for searching and composing 'best' sequences of actions in many domains [9]. Approaches based on AI planning are able to address the challenges of building environments and to overcome the drawbacks of rule-based approaches. In fact, AI planning naturally supports the evolution of environments in pervasive computing — it is easily adaptable and flexible approach [8]. The domain model is maintained and modified in an organised and conceptual way — the purpose of each action is always known. Furthermore, situations that require execution order among some or all actions in some coordination can be easily achieved. Finally, the same domain model can be suitable with minor modifications to a wide range of buildings.

Classical planning techniques require a set of actions, a goal and an initial state, and output a sequence of actions (or a plan) that, when executed in the initial state, achieve the goal. The basic correspondence with the problem of coordination in buildings can be established with actions, which correspond to service invocations, and the initial state, which corresponds to the current building context. A more interesting part is the specification of the goal. Since we expect that the service coordination will be initiated as early as occupant activities occur, the goal should correspond to the occupant activities in terms of requirements needed to be satisfied for the comfortable occupant performance. This means that to solve a planning problem, we need to identify the current occupant activities. Identifying activities relates to the problem of transforming low-level sensor data into high-level and meaningful information over which AI planning is well suited to reason. Fortunately, we can turn to human activity recognition to solve this problem. Occupant activities in buildings are generally well known and often limited in number, so we can use a predefined set of activities to recognise the occurring ones. In particular, we enforce ontology-based reasoning on sensor data to preprocess and classify the occupant activities [10]. Additionally, ontology-based activity recognition supports many other requirements specific to pervasive computing applications [11]. Once a set of activities is identified, we may establish a correspondence with the planning goal. A direct match can be established if each activity corresponds to some *procedure* without specifying details on what has to hold in final state for each activity. For support of procedural goals in AI planning, we can turn to Hierarchical Task Network (HTN) planning [12].

HTN planning is particularly useful due to its support for *rich domain knowledge* provided in task networks. For example, building standards can be easily encoded by using task networks. HTNs enable encoding information about how to perform some task or strategy, or how to accomplish something. This is convenient for services describing some environment-specific processes and strategies that involve other services. HTNs in fact allow encoding knowledge at varying levels of abstraction, focusing thus on a particular level at a time [13]. With this *modularity*, services of different granularity can be modelled as primitive and compound tasks. Furthermore, causal reasoning in more complex building domains can be easily lost if only planning actions are used. HTN planning through modularity can help by allowing device services to be encoded at the bottom level of the hierarchy. These services would then constitute the next more coarse-grained level. In addition, this *structured causality* supports and eases the evolution of building environments. If, for example, the evolution means a change in the building management system so as to fulfil new user requirements, then the causality of HTNs helps in determining which services have been affected by the change. Once such services are identified, the *flexibility* of HTNs makes it relatively easy to plug in new services as methods, or remove the obsolete ones without drastically affecting the hierarchy. Moreover, HTNs enable controlling the order in which services are executed, e.g., partial order, or methods evaluated (in the if-then-else manner). This provides a ground for modelling complex services with several control constructs, such as sequence, unordered, choice, if-then-else, etc. [13]. HTNs support recursion too. A compound task that is applied within its own definition is called *recursive*. In most cases, recursive tasks accomplish something by dividing it into smaller parts each of which is addressed with a recursive call to the task until reaching the base case. Recursive tasks aid the modelling of services that show a propensity for cycles, e.g., a service for turning off a collection of lamps. Finally, the knowledge encoded in HTNs helps in reducing the search space of planners and therefore finding a solution reasonably fast, if one exists. This makes HTN planners a "good compromise" between wide reusability and *effectiveness* in comparison to classical planners [14]. The latter planners do not require domain-specific knowledge, and thus they are widely applicable, however, they are characterised by weak efficiency. In pervasive computing, domains usually contain a large and constantly increasing number of services. HTN planning fits for such domains because it *scales* well to large number of tasks, and also generally to increasing size of planning problems [15].

We solve the problem of execution of services by using the concept of *orchestration* from service-oriented computing, which is the process of coordinating and executing services with the purpose to carry out the computed plans [16]. We propose an advanced orchestration model and a strategy to coordinate the receipt of events, the planing for new HTN planning problems, and the execution of their corresponding plans based on an effective and modern approach for plan orchestration in dynamic and uncertain environments [17]

With our solution, building management systems can anticipate occupant behaviour and provide dynamic and intelligent responses to different building situations. Recalling the example of lights in office buildings, a management system based on our solution will understand that Theodore is working with a computer and that the level of natural light for this activity is sufficient according to the building standard. It will thus not turn on the lights. Moreover, different compositions of services can be computed for the same objective depending on the current contextual state. Since the coordination of services is performed at runtime, building management systems can consider availability of services dynamically. Finally, our solution supports heterogeneous devices and is easily customisable to different building management systems.

Contributions

We summarise the contributions of the article next.

- We propose a novel solution to coordinating services in buildings while anticipating the occupant activities. The solution is built upon our preliminary effort [18], and, to the best of our knowledge, represents the first work that combines AI planning with activity recognition and provides a solution for the effective application of AI planning in buildings. Moreover, we establish a formal correspondence between the building coordination problem and an HTN planning problem. Being formalised, our solution provides the means for developing well-defined building management systems, allowing for improved maintainability, ability to evolve, reusability, consistency and sound reasoning [11].
- We extend our preliminary work by providing an orchestration model and a strategy to deal with inconsistencies during the execution of services.
- We design and implement our solution. The system is developed in the context of Energy Smart Offices (EnSO) (www.ensoffices.nl) and GreenMind Award (GMA) (www.sustainablebuildings.nl) projects. The system takes advantage of the service-oriented architecture model in the design of the software architecture. We implement all system components, starting from those responsible for devices providing services; components that store environment data and process it into context information together with activities recognised by the ontology-based technique; an HTN planner for composing services, and an orchestrator for dealing with plan execution and receipt of dynamic events.
- We report on the use of the system in two cases within Bernoulliborg, an office building at the University of Groningen in The Netherlands. First, we employed the system to control devices in two offices and a social corner at the fifth floor of Bernoulliborg given six activities of occupants. Second, we used the system to coordinate ceiling lamps given the presence and absence activities of restaurant occupants and the natural light level. We test and evaluate the system in terms of energy and monetary savings, usability, accuracy, and performance under increasing complexity. The results show that HTN planning together with activity recognition is a viable and beneficial solution to coordinating services in buildings.

The remainder of the article is organised as follows. Section 2 provides details on the building coordination problem, and Section 3 defines its correspondence with an HTN planning problem. Section 4 introduces the orchestration model and strategy. Section 5 gives insights into the architecture design and implementation of the system prototype. Section 6 illustrates the use of the system in Bernoulliborg, and presents the results of the several types of evaluations. Section 7 provides a discussion on our solution, system and results. Section 8 includes related work, and Section 9 finalises the article with concluding remarks.

2. Building coordination problem

Buildings are enriched with assorted devices, which report various kinds of data used to interpret the environments, some of which provide means to adjust environments to meet occupants' needs and operate efficiently at the same time. We say that a *physical model* of a building environment consists of a set of devices, where each device d has a sensing service s_d that returns the device's output (for a summary of notations used throughout the article, see Table B.5 in Appendix B). In Theodore's office, for example, there are two light sensors, one embedded near the window to sense the natural light level, and the other one placed above Theodore's desk to gather the indoor light level. The values of these sensors are returned through *getLux* services.

The set of outputs coming from devices could present pieces of data that has little meaning and may be conflicting. For example, if we assume that the two light sensors in Theodore's office are in fact distinct hardware components, it may happen that they sense contradicting values. In order to ensure a consistent view of the light conditions of the environment and to get meaningful information from the raw data, it is necessary to combine the outputs of several devices. We use a *data fusion* function to relate devices' outputs to variables, which represent an abstraction of the physical model. That is, $df : \{s_{d_1}, \ldots, s_{d_n}\} \rightarrow V$, where V is a set of building variables. Here we assume that the application of the data fusion function does not require assignments of values to all variables in V. We refer to this abstraction of the physical model as an *building environment*.

Definition 1 (*Building Environment*). A building environment E is a tuple $\langle V, L\rangle$, where V is a set of variables such that each v varies over a domain D^v, and L is a set of locations.

As a consequence, one or more unique variables can be associated with the office of Theodore. For example, *roomLux1* and *roomLux2* variables indicate the luminance level expressed in lux within the domain of, say, $[0, 1700]$. The association of variables with locations calls for spatial organisation of the building environment. Given a spatial property, we can associate each variable with the location to which it belongs, for example, *room564Lamp1* is at *desk*. In addition, the interrelationships over building locations can provide an abstract spatial representation about the building as a whole [19]. This representation can be used to define that Theodore's *desk* is a sub-location of *room564*. Generally, we may say that a *building property* is an expression of the form $prop(b, b')$, where $prop$ is the property name, b is either a building variable or location, and b' is either a location or a device type (see below).

When performing activities, occupants usually interact with the environment through its devices, causing changes in device outputs (e.g., changes in values of movement sensors). Occupant activities can be therefore identified and recognised via the building variables whose values are assigned on the basis of device outputs. In addition, activities and the location of their performing tend to be regular and repetitive in many types of building environments. In other words, we can derive that specific building locations are also associated with specific activities. For example, Theodore's office is a location where he works with or without his computer (PC) at his desk, or a meeting room where people typically give presentations or have discussions. We refer to this association as *activity area*: a logically defined space where some particular activity takes place involving one or more occupants [20].

Definition 2 (*Occupant Activity*)**.** An occupant activity act is a tuple $\langle n, l\rangle$, where n is the name of the activity, and $l \in L$ is the activity area.

For instance, $act = \langle \mathit{workingWithPC}, \mathit{room}\rangle$ is the activity of working with PC in the Theodore's office. In reality, there can be many activities happening in one location. The occurrence of such activities depends on the values of the building variables. This means that we can recognise all activities taking place at all locations in a building environment given the variables associated with each location. The ontological modelling of occupant activities and the ontology-based approach to activity recognition are described in Appendix A.1.

Definition 3 (*Activity Recognition*)**.** Activity recognition is a function $ar : E \rightarrow \mathit{Act}$, where Act is a set of activities.

Each $act \in \mathit{Act}$ is an occupant activity derived from the correlation of all $v \in V$ that are associated with the location of act. The set of recognised activities together with the abstraction of the physical model provide a snapshot of the building at a particular point in the time. We refer to such a snapshot as a *context*.

The context, through its variables, may not satisfy certain conditions of the building environment related to the performing of the recognised occupant activities. For example, if Theodore starts working with his PC, it may be that too much sun light is hitting his desk. For every building environment usually there is a set of quality conditions that the building must adhere to or maintain. Such conditions can be enforced by building standards, corporate policies, health protocols, etc., and can be related to activities with different concerns. In an office building, many activities and therefore the performance of occupants depend on the quality of light. For example, the European standard for lighting in indoor work places defines that basic requirements, such as light level, should be taken into account for existing and future buildings in general situations and diverse specific activities taking place there [6]. Conditions may specify a recommended state (e.g., the light level when working with a computer should be between 450 and 500 lux), or limit alterations (e.g., up to a certain amount of carbon can be emitted). The environmental perspective of such conditions encompasses not only the social dimension, such as the quality of life of occupants and health of people, but also the economic dimension, including resource management.

Since the satisfaction of such building conditions depends on building variables and their values, we can express a *building condition* ψ as a propositional formula over $(v = val) \mid (v \neq val) \mid (v < val) \mid (v > val) \mid (v \leq val) \mid (v \geq val)$ such that $v \in V$ and $val \in D^v$. For example, consider that when Theodore is working with PC in his office, the light level should be within the recommended range of 450 and 500 lux. Thus, the following condition must be satisfied and maintained by the building $(\mathit{room564PC} = \mathit{active}) \wedge (\mathit{room564Lux1} > 450) \wedge (\mathit{room564Lux1} < 500)$.

The occupant activities that have been recognised may require an immediate coordination of the environment so as the building can support the occupants in performing their activities as soon as possible. The upcoming coordination must be accomplished according to the building conditions and considering the building properties. In other words, given a building environment, its conditions and properties, and a *request* for a building coordination in the form of occupant activities, a building coordination problem consists of checking what has to be done in order to adapt the building environment according to the request.

Definition 4 (*Building Coordination Problem*)**.** A *building coordination problem* P^B is a tuple $\langle E, \Psi, \mathit{Prop}, \mathit{Act}\rangle$, where E is the building environment, Ψ is a set of building conditions, Prop is a set of building properties, and Act is a set of activities.

The physical model consists of a set of devices, and beside sensors, some of the devices represent also actuators, meaning they can act upon their values. We say that a device d, which is an actuator, has an *acting service* a_d that can change its value. For example, there are two actuators unobtrusively embedded in the lamps near by the door and windows of Theodore's office. These actuators are associated with an acting service that turns on lamps, and another service for turning off the lamps. In addition, each device representing an actuator may have an attribute that determines its type. For example, *room564Lamp1* is of type *ceilingLamp*. We represent device types using building property.

Given a set of acting services A, we say that α is a *satisfying coordination* for P^B if and only if $\alpha \subseteq A$ that transforms V into one that supports Act according to Ψ. In other words, a satisfying coordination refers to a sequence of acting services that changes the values of building variables such that the building supports the performance of occupant activities without violating building conditions. For example, turning off all lamps in Theodore's office is a satisfying coordination for the activity working with computer when the light level would be 480 lux, which is in accordance with the building conditions (between 450 and 500 lux.) A request, Act, is *achievable* if and only if there exists at least one satisfying coordination for it. This means that if there is no sequence of acting services that can support the performance of current occupant activities while satisfying building conditions, then the occupant activities cannot be supported as expected.

3. Correspondence to an HTN planning problem

We can now describe the correspondence between a building coordination problem and an HTN planning problem. For details on HTN planning, we refer to Appendix A.2 and [21]. Assume P^B to be a building coordination problem and $\mathcal{P}$ an HTN planning problem. Then, s_0 is the initial state *corresponding* to V, and consists of the following ingredients:

- A predicate p corresponding to a Boolean variable $v \in V$, only when v evaluates to true. For example, a predicate $turned(room564Lamp1)$ corresponds to the variable $room564Lamp1 = true$.
- A numerical variable $\bar{v}$ corresponding to a numerical variable $v \in V$ associated with a value $val \in D^v$.
- A corresponding predicate for each property of the environment, e.g., the predicate $in(room564Lamp1, room564)$ directly corresponds to the building property indicating the lamp $room564Lamp1$ is in the room $room564$.

The next component, tn_0, is the initial task network corresponding to Act. We create the corresponding tn_0 if and only if there exist $t \in T_n$ for all $act \in Act$ such that $t = n_{act}$ and $|\tau| = 1$. For example, consider that Act consists of a single activity $\langle presence, room564\rangle$. The initial task network corresponding to Act is $\langle\{presence(room564)\}, \emptyset\rangle$ given that a compound task with the name $presence(area)$ exists in the domain theory, where $area$ is a variable.

The last component of an HTN planning problem is the domain theory. We map an acting service to a primitive task, where preconditions model only variables that the service deals with in order to let the service be executable, and effects model how the variables are changed after the service execution. This means we do not annotate the corresponding primitive tasks with environment-specific knowledge. We say that a primitive task o *corresponds* to $a_d \in A$ if and only if $pn(o) = a_d$, $pre(o)$ is a logical expression over V, and $eff(o)$ is a conjunction of expressions over V. For example, the primitive task $\langle turn-on-lamp(lamp), \neg turned(lamp), turned(lamp)\rangle$ corresponds to the acting service *turnOnLamp*, which changes the truth value of a given *lamp* variable.

Thanks to HTN planning, we can present all environment-specific knowledge in compound tasks. This knowledge includes the building properties and building conditions. Generally, the environment-specific knowledge can also contain preferences of users, and may be used to empower users with a set of tasks that can be executed along with the automated coordination of the environment. In all cases, the environment-specific knowledge can be organised in some form of *complex services*. For example, a service to adjust the light level in a room. This service may have requirements involving building properties and building conditions, and several types of reactions that include acting services (or other complex services). For example, consider that the service is used for an area where working with PC is performed. The service involves different ways of adjusting the light given the current light level and constraints for this activity. The ways can be defined based on whether the light level is lower, within or higher than the recommended range (between 450 and 500 lux). Then, a compound task *corresponds* to a complex service, where the task's methods define the ways in which the service can react, and preconditions correspond to the service's requirements. The acting services and other complex services involved in the reactions compose methods' task networks.

We can now propose a correspondence between a building coordination problem and an HTN planning problem.

Proposition 1. *Let P^B be a building coordination problem and $\mathcal{P}$ be an* HTN *planning problem. $\mathcal{P}$ corresponds to P^B if and only if s_0 is the initial state corresponding to V; tn_0 is the initial task network corresponding to Act; $\mathcal{V} \subseteq V$; $\mathcal{Q}$ is the set of predicates corresponding to Boolean variables in V and $Prop$; and $\mathcal{T}$ is the set of tasks such that primitive tasks correspond to acting services in A and compound tasks correspond to activities in Act and complex services based on $Prop$ and Ψ.*

The following properties hold.

Theorem 1 (*Completeness*)**.** *Let P^B be a building coordination problem and $\mathcal{P}$ be the corresponding* HTN *planning problem. If a request Act is achievable, then there exist a plan π for $\mathcal{P}$.*

Proof. Assume that α is a satisfying coordination for P^B such that Act is achievable. From Proposition 1, there exist an HTN planning problem $\mathcal{P}$ corresponding to P^B. Since Act is achievable for P^B, there exist a plan for $\mathcal{P}$ that accomplishes tn_0. □

We can now obtain that the plan of the building-based HTN planning problem is a solution to the corresponding building coordination problem.

Theorem 2 (*Soundness*)**.** *Let P^B be a building coordination problem and $\mathcal{P}$ be the corresponding* HTN *planning problem. If there exist a plan π that is a solution to $\mathcal{P}$, then we can construct a sequence of acting services α based on π that is a satisfying coordination for P^B.*

Proof. Assume that there exist a plan π for $\mathcal{P}$. Since $\mathcal{P}$ corresponds to P^B, we can map each primitive task from π to an acting service from A. Given that π is a solution to $\mathcal{P}$, it means that Act is achievable for P^B according to Theorem 1. Thus, the resulting sequence of acting services is a satisfying coordination for P^B. □

For example, let P^B be some building coordination problem for which the corresponding building-based HTN planning problem is $\mathcal{P}$. Furthermore, consider the following plan for $\mathcal{P}$: [*turn-on-lamp* (*room*564*Lamp*1), *turn-on-lamp* (*room*564*Lamp*2), *turn-on-monitor* (*room*564*Monitor*1), *turn-off-lamp* (*room*432*Lamp*1)]. We can construct a sequence of acting services where services are mapped from the plan actions. The resulting sequence of acting services is a satisfying coordination for P^B.

The above formalism affirms our intention to solve a particular problem in pervasive computing. The correspondence enables a straightforward mapping from the building coordination problem to the HTN planning problem that is actually solved. It allows for the use of plans as sound solutions to the building coordination problem. Moreover, the formalism is not limited to a specific building environment, but provides an abstraction of building environments. Thus, it can be leveraged to employ many and different implementations of pervasive computing systems. Finally, it can help in improving the maintainability and evolvability of such systems [22].

4. Orchestration

We have now established a correspondence between building coordination problems and HTN planning. We still have to bring the dynamics of buildings into perspective, that is, events happening continuously during plan execution.

Many studies try to interleave planning with the execution of plans and context changes by continuously revising already computed plans so as to achieve the given goal [19]. Although such an interleaving may be a more or less suitable approach, depending on the nature of the adopted planning technique, it commonly causes performance deterioration to the underlying system. This is usually due to the time spent on revising a plan, and even more, ending up planning from scratch when the revision is unsuccessful. A simpler approach, e.g., always planning when some context update or service failure is encountered, brings a trade-off between spending unknown time on revising and/or planning from scratch and accepting a computation time with a known upper bound — the one of the planning step.

Such an approach still has some challenges to be overcome, such as dealing with continuous events, consistent updates of the planning state, continuous execution of plans for the respective HTN planning problems, and concurrency. We refer to the process of receiving events, creating HTN planning problems, and executing the corresponding plans as *orchestration*.

4.1. Model

Our orchestration model is a variation of the one from [23]. The main difference between the models is that ours does not deal with observations. The orchestration relies on the concept of a state model [24], which includes (environment) states, acting services, and a state-transition function that defines the mapping from one state to other. In addition, the orchestration involves events, meaning that an update function is needed to change the values of variables with respect to assignments provided in the events. We assume that the assignments in events refer to different variables.

Definition 5 (*Orchestration Model*)**.** Let E be an environment. An orchestration model Σ based on E is a tuple $\langle \mathcal{S}, A, \mathcal{E}, \gamma \rangle$, where

- $\mathcal{S}$ is a set of states,
- A is a set of acting services,
- $\mathcal{E}$ is a set of events. An event assigns a value *val* to a variable $v \in V$ such that $val \in D^v$,
- $\gamma : \mathcal{S} \times A \times \mathcal{E} \rightarrow \mathcal{P}(\mathcal{S})$ is a transition function $\gamma(s, a, \epsilon) = \{\delta(s', \epsilon) \mid s' = s[a]\}$, where $\delta : s \times \mathcal{E} \rightarrow \mathcal{S}$ is a function that updates a state s using the assignments from $\mathcal{E}$ and $s[a]$ is as defined in Appendix A.2.

The orchestration model is the one determining the orchestration. In other words, an orchestration ω refers to a sequence of pairs of events and plans $\langle(\epsilon_0, \pi_0), \ldots, (\epsilon_k, \pi_k)\rangle$. The orchestration problem then concerns maintaining a consistent planning state given uncontrolled events, and finding and executing a sequence of acting services that satisfy some request given such a state.

Definition 6 (*Orchestration Problem*)**.** An orchestration problem P^O is a triple $\langle \Sigma, V, Act \rangle$, where Σ is an orchestration model, V is a set of building variables, and Act is a set of activities.

Upon each context change, an HTN planning problem is created with a state resulting from the application of δ. Thus, for a sequence of events $\epsilon_0, \ldots, \epsilon_k$, there is a sequence of HTN planning problems $\mathcal{P}_0, \ldots, \mathcal{P}_k$ and their corresponding plans $\pi_0, \ldots, \pi_k$. We refer to the sequence of plans $\langle \pi_0, \ldots, \pi_k \rangle$ as an orchestration plan Π. Since events may come fast in a sequence, many of these plans will contain redundant acting services. On the other hand, parts of some existing plans may be already executed before a new plan is set for execution and whose acting services may contain a subset of already executed ones. In both cases, we can reduce Π by removing the redundant acting services without introducing conflicts among dependent services. A reduced orchestration plan Π' is a sequence of modified plans $\langle \pi'_0, \ldots, \pi'_k \rangle$ such that it is a correct execution [21], that is, it still achieves Act. Then, Π' is a solution to P^O if Π' is part of an orchestration ω that produces a sequence of states such that achieves Act.

4.2. Algorithm

We design an algorithm that supports concurrent processing of incoming events and executing actions. In this way, the orchestration can process messages coming in parallel from the changes in the environment and service invocations, and will not get blocked by acting services unable to react on the first invocation. The algorithm is able to handle simple service failures and tolerate events that may affect the orchestration model. Thus, the algorithm takes a pragmatic attitude and refers to planning whenever an event is received.

The algorithm is shown in Algorithm 1. The RECEIVE function indicates that messages are received asynchronously. A message informs the algorithm that a building environment has to be created. This happens only when the algorithm is initiated, and involves gathering all information relevant for the environment, such as variables, locations, device types, and properties. Given all these ingredients, the algorithm coordinates the creation of corresponding HTN planning constructs. We assume that only one construct, the domain theory, is encoded in advance and made available for the orchestration to retrieve it using a unique domain name (line 16). Then, the domain object is stored and available for all planning requests. The state is created dynamically with respect to V and building properties (line 18). When a planning request is received, the orchestration invokes the planner given the current HTN planning problem $\mathcal{P}$ (line 22). For every event received from the environment (line 8), if ϵ represents a new value for a variable, either a predicate or numerical variable is updated in the state. Otherwise, the event is added to the network of tasks to be accomplished. Upon each event, independently of its type, a planning request is issued to find a new adaptation for the environment. When a plan is received for execution (line 28), first the plan tasks are mapped into a set of acting services. If possible, some of these services are executed in parallel. When an acting service completes its execution, the orchestration checks the response. When a service fails, the orchestration re-executes it for a fixed number of times. If this step is unsuccessful, the failure is permanent and the algorithm removes the service from the reduced orchestration plan and issues a planning request.

Algorithm 1 Orchestrating incoming events and execution of corresponding plans

```
 1: $E$, $\mathcal{D}$, $s$, $\Pi'$ empty
 2: $tn \leftarrow$ default task
 3: function RECEIVE
 4:     case InitialiseEnvironment():
 5:         Gather all information about the environment $E$
 6:         Create corresponding planning constructs based on $E$
 8:     case ContextChange($\epsilon$):
 9:         if $\epsilon$ is an activity then
10:             Update $tn$
11:         else
12:             $p \leftarrow$ CONVERTTOPREDICATE($\epsilon.v$,$\epsilon.val$)
13:             UPDATESTATE($p$)
14:         Plan for the latest $tn$ and $s$
16:     case ConstructDomain($domainName$):
17:         $D \leftarrow$ CREATEDOMAIN($domainName$)
19:     case ConstructState():
20:         $s \leftarrow$ TOSTATE($E$)
22:     case PlanReq($taskNetwork$):
23:         $P \leftarrow \langle D, s, taskNetwork\rangle$
24:         $\pi \leftarrow$SEARCH($P$)
25:         if $\pi \neq \emptyset$ then
26:             Execute $\pi$
28:     case ExecutePlan($plan$):
29:         $A \leftarrow$ CONVERTTOSERVICES($plan$)
30:         Add $A$ to $\Pi$
31:         for $a_d \leftarrow A$ do
32:             $a_d$ completes
33:             case $a_d$ is failure:
34:                 Retry the service $n$ times
35:                 if $a_d$ has failed $n$ times then
36:                     Remove $a_d$ from $\Pi'$
37:                     Plan for the same $tn$ and current $s$
39:             case $a_d$ is success:
40:                 Remove $a_d$ from $\Pi'$
41:                 Check whether $s$ is consistent with the effects of $a_d$
43:         end for
45: end function
```

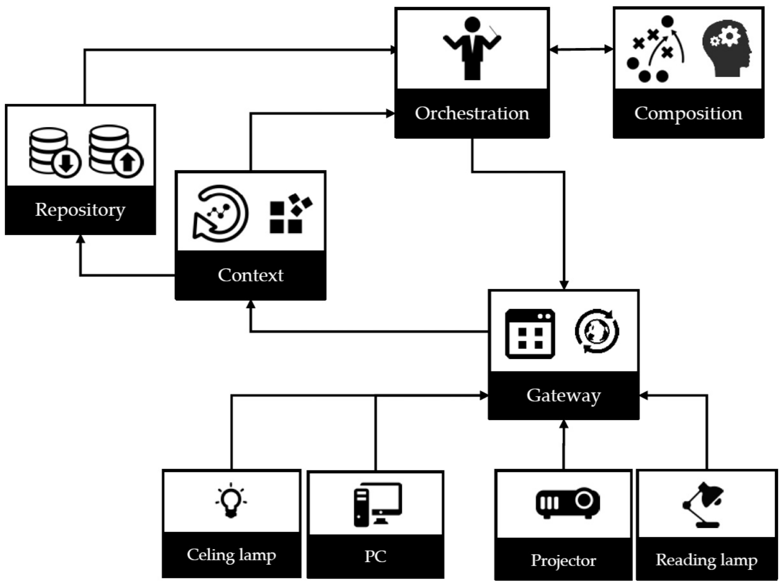


Fig. 1. High-level overview of the architecture design.

5. Design and implementation

With the orchestration, we can complete a whole operation cycle of buildings' operation, starting from the collection of data through sensors; processing it into context information; composing acting services to coordinate the building environment by anticipating the occupants activities; until the execution of acting services upon devices representing actuators. Next, we provide general considerations for the architecture design, and details of the prototype system.

5.1. Design

Fig. 1 shows a high-level overview of the architecture design, including the main components and interaction among them. Devices of any type reside at the bottom of the diagram. Each device has an interface that provides an access to device's data and control. The Gateway component is the point where a unified way of interaction with the devices is provided — device functionalities are offered as services to interested components. The Context component is where the data arriving from devices is accepted, and the state of each device is continuously monitored. The data is collected, aggregated into meaningful and consistent information, and provided to interested components. All building data is stored into the Repository component. The remaining two components are responsible for the orchestration and composition of services.

We consider the interaction between the components to be based on message queues, that is, the components use queues to communicate with each other with or without knowing the physical location of other components. Message queues can significantly simplify the implementation of the separate components and also improve the performance, scalability and reliability of the system. Also, with message queues, senders and receivers of messages do not need to communicate at the same time as the messages are stored onto the queues until the recipients receive the messages. As a result, the communication between the components can be asynchronous.

5.1.1. Devices and gateway

Devices form sensor networks and provide a basic infrastructure for gathering raw contextual data from the building environment. We associate the devices with so-called Hardware Interfaces (HIs) to gather data and control them. For example, we can gather data about the ambient context from light and temperature sensors by using their HIs.

The Gateway offers an application service that manages the devices deployed in the building environment. This service is responsible for interfacing with sensor networks that use different protocols. It serves as a protocol translator as well as fault isolator. The diversity of sensors implies that the system should be able to address the heterogeneity of sensor networks. The Gateway service is thus capable of managing and integrating different brands of devices into the system. The Gateway service reads information from various networks and join the data read from them into only one standardised message with a specific format. It also provides an interface for upper layer components. In addition, the Gateway encapsulates the complexity of sensor networks and supports the necessary abstractions of physical devices.

5.1.2. Context

Context services include context processing (CP) and activity recognition (AR) services. The Context services provide the system with the ability to process the sensor data received from the Gateway. The outcome of CP and AR services is clean, consistent, enriched, and high-level information about the building environment. Such information can in turn be used by the Orchestration.

5.1.3. Repository

The Repository offers services to store and retrieve data about a building environment. There are mainly two types of building information. One involves descriptions of devices deployed in the building, their types and locations within the environment, their data type and value ranges, the layout of the building, etc. The other type of information is dynamic and involves the data coming periodically from devices directly or from the Context. While the data can be stored in any type of databases (e.g., relational databases, NoSQL, graph-based databases), the database choice is of great importance for the performance of the Repository. For example, graph databases support index-free adjacency that can lead to a much better performance as compared to relational databases. In addition, each sensor reading has relationships to other data, such as the floor, time and date, and room or user. While graph databases have advantages for maintenance of building structures and relationships between entities (e.g., users, rooms, devices), NoSQL databases may deliver faster performance for systems involving time series or data coming from thousands of sensors.

5.1.4. Orchestration and composition

The Composition has several planning services organised in three classes: (1) modelling services, which provide capabilities that focus on the planning entities used to form an HTN planning problem; (2) problem-solving services, which have capabilities bounded within the process of solving planning problems; and (3) utility services, which include capabilities related to message conversion, storage of domain models, communication with other components, exception handling, etc.

Since building environments are expected to support continuous orchestration, we need to adopt a strategy that enables long-running runtime activities of the Orchestration component. The Orchestration receives events from the Context continuously and reacts immediately and accordingly. When necessary, the Orchestration uses the planning services to create an HTN planning problem and to compute a plan. If plan is found, the Orchestration executes it using the Gateway services.

5.2. Implementation

We provide insights into the implementation of each component of the architecture. All components communicate with each other using JSON (www.json.org). We opt for JSON due to its good readability, simple syntax and ease of use. For the sake of traceability with asynchronous communication and readability and consistency in generic messages, a basic template is specified to which all the messages of components should adhere. Depending on the data that a component requires, the body of the JSON object may change. The RabbitMQ [25] messaging framework is chosen as it is a highly reliable enterprise messaging system based on the emerging AMQP (www.amqp.org) standard and runs on all major operating systems.

5.2.1. Devices and gateway

We employ wireless TelosB-based sensors compliant with the IEEE 802.15.4 standard and produced by Advantic Systems (www.advanticsys.com). On-board passive infrared sensor (PIR) based movement detectors, microphone and FlexiForce pressure sensors are used together with a light sensor (Hamamatsu S1087 Series). The motes are programmed in nesC and run on the TinyOS 2.1.1 embedded operating systems (www.tinyos.net). As power meters, we use Plugwise (www.plugwise.com) products consisting of plug-in adapters that fit between a device and the power socket (see Fig. 2). The adapters can measure the real-time power consumption of the device plugged in, and can turn on and off the device at the same time. The adapters form a wireless ZigBee (www.zigbee.org) mesh network around a coordinator. The network communicates with the base station through a link provided by a receiver device (in the form of a USB stick device).

We use the Gateway services to transform low-level and device-specific communications into JSON objects which our system understands, and to provide an interface to interact with the low-level commands of the devices. We implement two Gateway services, namely *TelosB service* and *Plugwise service*. The TelosB service handles the readings from the TelosB sensors. The Plugwise service handles the readings from the Plugwise adapters, and provides an interface to interact with the devices plugged in. The data which the service reads includes the real-time consumption and state of the devices. The interface allows other components to remotely change the state of a device plugged in a Plugwise adapter. The Gateway services are implemented in the Scala programming language (www.scala-lang.org), and run on an Intel NUC thin client device (www.intel.com), which is deployed in the restaurant itself.

5.2.2. Context

The raw sensor data about the environment provided by the Gateway services is used as input for the Context services. We implement the two Context services: CP service and AR service. CP is responsible for ensuring a consistent view over the

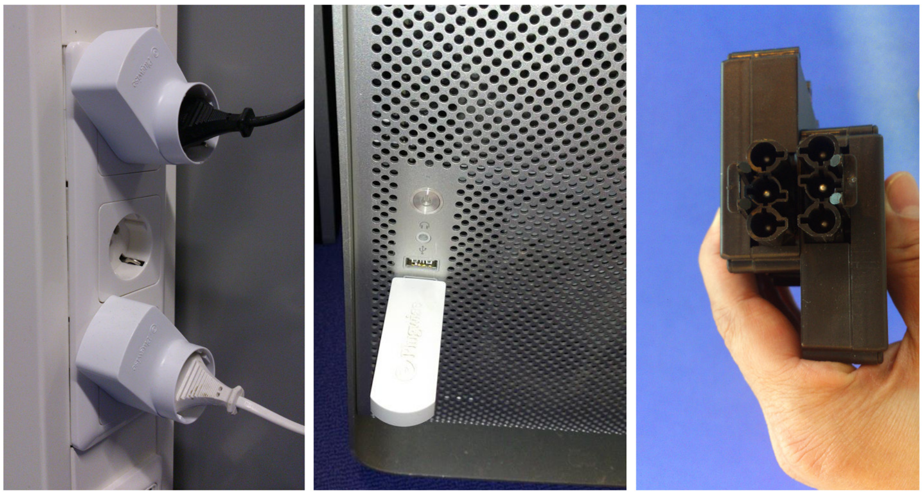


Fig. 2. Power meters and a receiver device.

Table 1
Description of activities considered for the evaluation of our system.

Activities	Definition	Involved artefacts
Working with PC	Person uses PC, PC and screen are turned on, keyboard/mouse are manipulated	Chair, PC, screen, keyboard/mouse
Working without PC	Person sits at a desk, PC and screen are turned off	Chair
Having meeting	Two or more persons discuss	Chairs, human voice detector, also PC
Presence	Person is active in an area but no further specific activity is recognised	Movement detector
Absence	User is absent from an area	No artefacts involved
Having coffee	Persons have lunch/coffee or just a break	Microwave, movement detector

ambient environmental condition, such as the natural light level. That is, raw sensor readings, coming from a light sensor in the unit of lux, are calibrated and correlated in a standard form for the light condition at different locations. To calibrate and correlate the readings, we firstly collect historical data from the sensors associated with the ground truth about the natural light levels. Then, the CP uses this information to classify the incoming raw sensor readings into appropriate natural light conditions.

The AR service is responsible for processing the data related to occupant activities. For example, the movement data coming from PIR sensors is used to reason over for the presence activity. The occupant presence of an area can be determined from one or many PIR sensors, depending on the pre-configuration of the environment. Table 1 shows the definitions and involved artefacts of the activities considered for the evaluation of our system.

We also implement a mechanism that adapts the timeout of the presence activity depending on the time of the day. For example, in the Bernoulliborg restaurant, in the early morning, when the chance of occupants staying long in some area is low, the timeout is set as low as 5 min, while during the lunch time, when occupants would stay long for their lunch, the timeout is set to 30 min. The time values are adjusted based on observations.

5.2.3. *Repository*

We use two different types of databases for the storage of data, namely Neo4j (www.neo4j.com) and Cassandra (www.cassandra.apache.org). Neo4j is an open-source graph database with features from both document and graph database systems and excels in scalability, availability, performance, and price. We use Neo4j as a storage facilitator for the environment's spatial information. The Neo4j database contains static information about the building, which includes information about sensors and actuators, their location, their device type and the type of data they produce.

Cassandra is a NoSQL database that delivers fast performance for systems involving time series or big data. We use Cassandra to store time series data coming from sensor readings. This allows us to record and review the behaviour of the environment at any given moment in time.

5.2.4. *Composition*

We illustrate how to specify a domain theory for a building environment in Hierarchical Planning Definition Language (HPDL) [26]. For details on the syntax for HPDL based on extended Backus Naur Form, we refer to [27]. The problem specification, including the initial state and initial task network, is created with respect to the elements defined in the domain specification. The HPDL domain and problem specifications constitute an HTN planning problem which is provided as input to the Composition component represented by the HTN planner.

Domain model. We encode device types and locations as domain types, which are all subtypes of the type `object`. For example, consider that the restaurant has two types of lamps: security and regular lamps. Then, `regularLamp` and `securityLamp` are domain types whose supertype is `lamp`, which is a subtype of `object`.

Environment properties are specified as predicates of the form `(prop arg1 - type1 arg2 - type2)`. An example of a location-related property is `(in ?a - area ?r - room)`, which denotes that an area is within some room.

Boolean variables are encoded as predicates too. Such a predicate is in the state only when the corresponding variable evaluates to true. The predicate name indicates the truth value, and the predicate's only argument is the variable itself. An example of such a predicate is `(turned ?l - lamp)`.

Numeric variables are modelled as domain functions. A domain function returns the value of the variable it represents. The function name describes the sensor associated with the variable (we get this association from the Neo4j database). A function may also have arguments denoting variables to which the numeric variable is related. For example, `(light-level ?a - area)` encodes the variable representing a natural light sensor deployed within some area.

Acting services are encoded as primitive tasks or actions in HPDL terms. We use a parametrised action for the same type of acting services. Then, a parameter denotes the actual variable we want to act upon. In this way, we ensure the uniqueness of each action instance. An action may have other parameters that refer to the input variables of the corresponding acting service. The preconditions and effects capture the semantics of the service. Since we have acting services without semantic annotations, the actions are very simple. The HPDL action in the following represents an acting service for turning on a lamp only when the lamp is turned off.

```
(:action turn-on-lamp
  :parameters (?l - lamp)
  :precondition (not (turned ?l))
  :effect (turned ?l))
```

Complex services are modelled in compound tasks. Let us assume that we need to adjust the light level in some area. There are two possibilities for this: to increase or decrease the light level. These two possibilities indicate two complex services that require different modelling but similar reasoning. Thus, each complex service is a compound task, namely `increase-level` and `decrease-level`. Each method encodes a specific way of accomplishing the respective task. Let us focus on the `increase-level` task. If there is not enough light, it means that we need to turn on some lamps. While doing that, we need to ensure a certain level of light according to the European standard for lighting in indoor work places. We consider the light requirements for each type of space as prescribed by the standard. Then, one method of the task deals with the case when there is deficient light and there are lamps that can be turned on. That is, if the estimated light level of the given area is not within the recommended light-level range, and there are lamps that can be controlled and are turned off, then we can turn on a lamp, estimate its effects on the light level (simulated sensing), and recursively call `increase-level`. The following encoding represents this case.

```
(:method deficient-light-level
  :precondition (and (>= (light-level ?a) ?x) (< (light-level ?a) ?y)
                     (in ?l ?y) (regularLamp ?l) (not (turned ?l)))
  :tasks (sequence (turn-on-lamp ?l) (estimate-light-increase ?l)(increase-level ?a)))
```

The process of estimating the effect of a lamp on the light level is encoded in a separate task. This is necessary because, first, we keep primitive tasks simple, which means they do not perform sensing, and second, we perform off-line planning with completely instantiated variables, and thus, deterministic outcomes of actions.

Another method deals with the case when all lamps are already turned on. To address all of the lamps, for example, in the area of our interest and in the areas near by our area, we use a *forall* expression.

We model user activities as separate compound tasks. These tasks further involve other compound tasks to ensure a satisfactory state of the building, resulting in a hierarchy of tasks as defined in Definition 7. For example, consider the case of a presence activity in some area. The corresponding compound task involves `increase-level` and `decrease-level` tasks, depending on the estimated level of light given the information in the current state. The task encoding the absence activity, which is demonstrated in the following, is simpler and it indicates that we can turn off all lamps associated with the area of interest. However, this does not mean that the area's lamps cannot be turned on later during planning. If areas, which are near by this area, have presence activities and insufficient light level, then it means that some lamps of this area will be turned on despite its absence activity. This is taken into account in the hierarchy of the `increase-level` task. The modelling of other activities can be realised using reasoning analogous to the one described so far.

```
(:task absence
  :parameters (?a - area)
  (:method turn-off-all-lamps
   :precondition ()
   :tasks (sequence (turn-off-lamps ?a))))
```

Planner. We use our own HTN planner, called Scalable Hierarchical (**SH**) planning system, implemented entirely in Scala. **SH** consists of several subcomponents, namely the Problem converter, HPDL processor, and Planner. The information coming from the building environment is transformed into HPDL problem specification through the Problem converter. We design the converter upon the Cake pattern [28], which enables to solve dependencies between components through abstraction of implementation details. In particular, we define an interface for the Problem converter through a Scala trait that may be instantiated in multiple ways depending on the number and types of environments we need to implement.

We assume that a domain model specified in HPDL is predefined. The HPDL problem and domain descriptions are then transformed into programming-level constructs through the HPDL processor. This processor uses parser combinators to transform HPDL descriptions into Scala objects. Parser combinators enable efficient processing of strings by combining parsers into a new parser [29].

We provide planning as a service by implementing **SH**'s capabilities as Representational State Transfer (REST) resources (Web services) [30]. Upon receiving a request with appropriate arguments, **SH** may check, for example, the correctness of an HPDL domain or problem, the consistency of a required problem and domain, or search for a solution. The planner replies to an interested component with an appropriate to a situation answer. For example, **SH** may provide a plan in JSON format, or it may show a syntax failure at a specific position in the domain encoding.

5.2.5. *Orchestrator*

Since the orchestration algorithm is stateful, i.e., it maintains a model of the environment, including the planning state, domain, task network and reduced orchestration plan; and, in order to support our design assumption (i.e., concurrent use and updates of the state), we built the algorithm upon the Actor model [31]. It receives messages asynchronously and reacts to them by making local decisions, creating other actors to handle specific and/or concurrent messages, sending new messages, and deciding what to do upon the next message received.

The orchestration algorithm is implemented in Scala. We refer to the implementation of the algorithm as *orchestrator*. Instead on concrete implementations, the orchestrator depends on abstractions of other components. To accept different types of environments, the orchestrator uses a trait called Environment, which is specified upon Definition 1. In our case, the environment is represented by an implementation of the restaurant. The orchestrator populates a specific environment by retrieving the information from the databases. The set of variables, their types, locations, and properties are gathered from Neo4j. The initial values of variables are gathered from Cassandra. Then, the orchestrator subscribes to RabbitMQ, and awaits for messages, that is, events.

The orchestrator creates a domain object with the help of **SH** planning services, and uses the Problem converter to transform Environment in a planning state as described in Section 3. Upon each event, the orchestrator creates a corresponding HTN planning problem and invokes the core planning service of **SH**. When a plan is found, if such exists, the orchestrator translates the plan steps into acting services and uses the device services implemented as REST resources for execution. The orchestrator itself is built as a container for the Docker platform (www.docker.com), which automates the process of deployment of distributed applications.

6. Deployment and evaluation

We deploy the system in our own building, Bernoulliborg (53° 14′26.1″N 6° 32′11.1″E), at the University of Groningen, The Netherlands. Bernoulliborg is an office building of more than 10000 m^2 erected in 2008.[1] The number of staff working there is 307 and the capacity of students that can attend lectures is 500. The building has 180 offices, 16 lecture rooms, 8 meeting rooms, 6 social corners, and a restaurant, which is an open space located on the ground floor. We show the use of our system in two cases. For the first case, we exploit two offices and one social corner together with the experimental results as presented in our preliminary work [18]. For the second case, we use the restaurant as an environment for the deployment and testing of our system.

6.1. *The case of offices and social corner in Bernoulliborg*

The two offices and the social corner cover a total area of 60 m^2. Each space has several small and large windows from one side or two sides, letting a large amount of natural light to come into the spaces. The offices are occupied by two Ph.D. candidates, while the social corner is used by all staff at any time.

The layout of the offices and social corner together with the network of power meters attached to appliances and simple sensors is illustrated in Fig. 3. In particular, each space has three controllable light fixtures (or lamps). The lamps are T8 fluorescent Philips double tubes. The lamps consume 32 W each. The lamps are equipped with the Plugwise plugs, which make the lamps controllable but also provide information about the power consumption of the fixtures.

We consider simulated control of the workstations of the Ph.D. candidates in the two offices. A workstation consists of a standard personal computer (PC) with an LCD 24″ screen with consumption when active of 100 Wh and 20 Wh, respectively. Thus, the total electricity consumption of a workstation when being actively used is 120 Wh, and only 5 W/h while in sleep mode. We attach power meters to each PC and screen to detect the power state and measure the power consumption. In addition, we take into account the use of a microwave at the social corner. The microwave is associated with a power meter to measure its current power consumption. We installed 14 more sensors to estimate chair's occupancy, to measure the

[1] www.nl.wikipedia.org/wiki/Bernoulliborg.

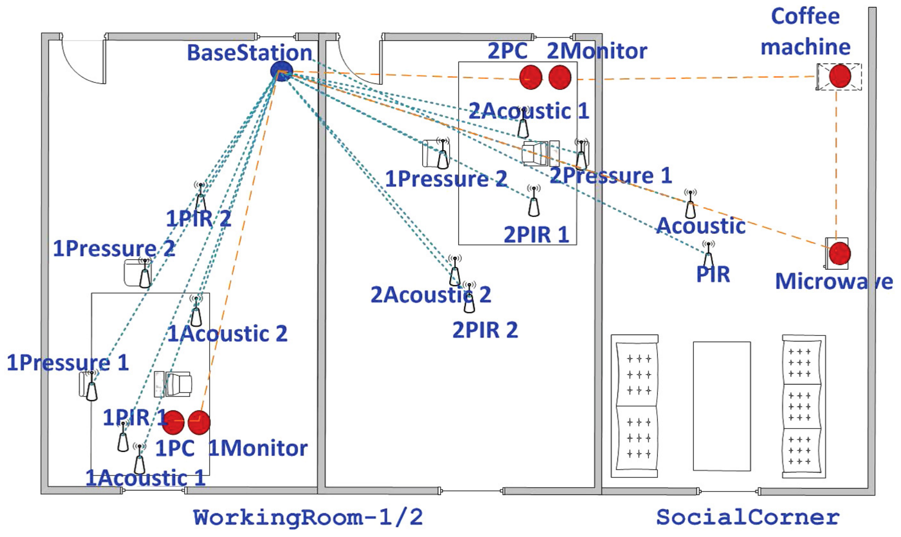


Fig. 3. Schematic representation of the offices, social corner, and deployed devices.

human voice level, and to detect people's movement. Finally, we simulate a natural light sensor by using weather conditions retrieved from Freemeteo [32]. Namely, clear weather indicates 700 lux, few clouds 550 lux, partly cloudy skies 500 lux, cloudy skies 425 lux, light snow 400 lux, and mist indicates 350 lux.

We conducted tests on the system during the working hours of three days. We analyse two scenarios to show the potential for energy saving. In the first scenario, we consider *regular control* where occupants control workstations and lamps manually. In the second scenario, we consider automated control where the planner takes into account both the natural light level and recognised occupant activities to control the workstations and lamps automatically. Here we do not actually control the workstations, but we assume the existence of a system based on a client–server paradigm that enables bringing workstations into sleep mode or waking them up on planner's demand, e.g., [33].

6.1.1. *Energy savings*

For the regular control, we consider that the two workstations and lamps at all three areas are turned on for the whole working time given the fact is it common office workers to leave their computers on even when they are not using them and no power management options are enabled [34–37]. The average consumption is then 0.528 kWh.

For the automated control, the workstations are in active mode only if *working with PC* is taking place, otherwise they are brought to sleep mode. Lamps are controlled in accordance with the European standard on lighting in indoor work places. In other words, for working with PC, having a meeting and being present require low levels of light ($\sim$500 lux), while working without PC requires a light level of 750 lux.

Our system provides an intelligent coordination of the use of workstations and lamps with respect to the activities of occupants and the natural light level. Fig. 4 illustrates the electricity consumption of both the regular control and our system at every minute. Significant reduction of electricity consumption is evident in the case of automated control due to the consideration of the natural light level, thus turning as many lamps as possible, and brining workstations into sleep mode as soon as occupants do not work with PC. In particular, no lamps were needed on the first day, while on the second and third day only five lamps were turned on for 11 min and 24 min, respectively. Similarly, two workstations were needed in total for 5.13 h on the first day, 8.47 h on the second day, and 8.27 h on the third day.

Table 2 shows the potentials for energy saving of our system. When compared to the regular control, the combination of activity recognition with planning brings a significant potential for energy reduction of 75.5%, including 46.9% and 98.5% savings from controlling the workstations and lamps, respectively.

6.1.2. *System performance*

We are interested in the accuracy of the system. The system is accurate if it is able to precisely recognise and classify occupant activities.

During the tests in the three separate days, the system recognised six different activities performed by the two Ph.D. candidates in the two offices and social corner. Using pen and paper, the occupants took accurate notes of the actual activities happening every minute, which are then used as golden truth for the evaluation. Table 3 shows the real occurrences of activity instances at each area. We collected ground truth composed of 1201 activity instances by using one minute time extent. The overall success rates of the activity recognition service for *WorkingRoom* — 1, *WorkingRoom* — 2 and *SocialCorner* are 80.85%, 76.35% and 99.17%, respectively. On the other hand, the total number of minutes of wrong recognition for

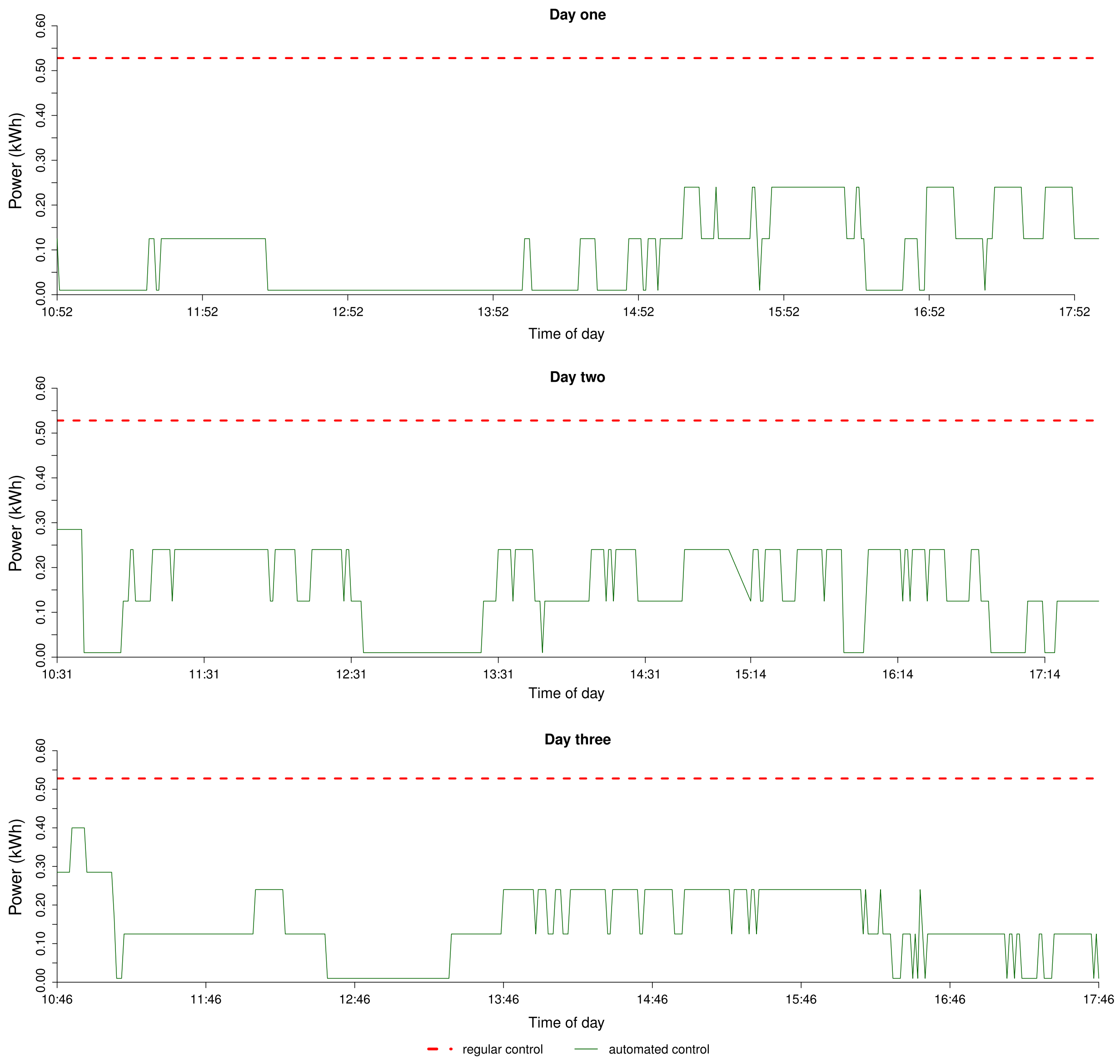


Fig. 4. Comparison of electricity consumption between regular control and our system.

WorkingRoom — 1, *WorkingRoom* — 2 and *SocialCorner* is 230, 284 and 56, respectively. A detailed analysis of these results is presented in [38].

6.2. *The case of restaurant in Bernoulliborg*

Photos from the restaurant are shown in Fig. 5. It covers a total area of 251,50 m^2 with a capacity of 200 sitting places. The restaurant has glass walls from three sides, enabling a significant amount of natural light to come through when the weather conditions allow for it. The restaurant area is used for lunch in the period from 11:30 a.m. until 2:00 p.m. Outside these hours, the area is used by staff, students or other visitors for working, meeting, or other purposes.

The restaurant area is an open space divided in two sections by construction. We make use of this division in our set-up. The layout together with the locations of deployed sensors and power meters is illustrated in Fig. 6. In particular, each section has 15 controllable light fixtures (or lamps), making 30 in total. All lamps are fluorescent Philips tubes. There are several light fixtures that are uncontrollable and represent security lamps. While we do not control these, we take into account the light that they provide. In addition, there are two types of controllable lamps. The first type are large lamps that have 38 W of power consumption each, and the second one are small lamps, each of which has 18 W. These lamps are controllable via the Plugwise plugs, which also provide information about the fixture's power consumption. We installed 15 more sensors,

Table 2
Potentials for electricity savings.

Date	Device	Total time (hours)		Consumption (kWh)	
		Regular control	Automated control	Regular control	Automated control
Day one	Lamps	64.98	0	2.08	0
	Workstation active	14.44	5.13	1.73	0.62
	Workstation sleep	0	9.23	0	0.05
Day two	Lamps	64.53	0.92	2.06	0.03
	Workstation active	14.34	8.45	1.72	1.01
	Workstation sleep	0	5.5	0	0.03
Day three	Lamps	63	2	2.02	0.06
	Workstation active	14	8.27	1.68	0.99
	Workstation sleep	0	5.82	0	0.03
Total electricity consumption of lamps				**6.16**	**0.09**
Total electricity consumption of workstations				**5.13**	**2.2**
Total electricity consumption				**11.29**	**2.82**
Electricity saved from lamps				**6.07kWh** (**98.5**%)	
Electricity saved from workstations				**2.41kWh** (**46.9**%)	
Total electricity saved				**8.48kWh** (**75.05**%)	

Table 3
Ground truth of the occurrences of activity instances at each area.

Activity	Area		
	WorkingRoom−1	*WorkingRoom*−2	*SocialCorner*
Absence	70	129	843
Presence	232	112	0
Working without PC	256	139	0
Working with PC	640	754	358
Having meeting	3	67	0
Having coffee	0	0	0

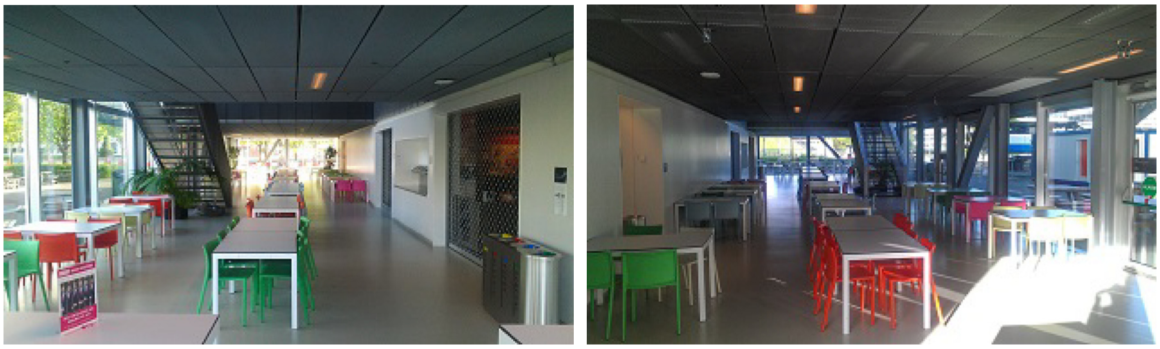


Fig. 5. Overview of the restaurant from the east and west sides.

one to measure the natural light level, and the rest to detect people's movement. In order to make a more meaningful use of the restaurant space given the movement sensors, we divide each section into smaller spaces, called *areas*. The areas are not necessarily of the same size, and we embedded movement sensors in each area in positions that cover most of the space of the respective area.

We conducted tests on the system over the course of five weeks in the months of February, March, and April 2015, involving measurements from Monday to Sunday. In the Netherlands, these months are dark months, meaning that we can experience some of the worst possible conditions to save energy. In the last three weeks in February, we recorded measures of energy consumption of lamps in order to understand the typical behaviour of *manual control* of lamps in the restaurant. This enables us to define a baseline. In the last week of March and first week of April, we allowed for *automated control* of the environment by using our system. Thus, manual control was disabled and the system was running continuously without interruptions during these two weeks. For the purpose of comparison, we simulate control of lamps based only on the information coming from the *movement sensors* during the period of automated control. We use the same organisation of sensors and lamps as in the case of automated control. We consider two types of activities, namely presence and absence.

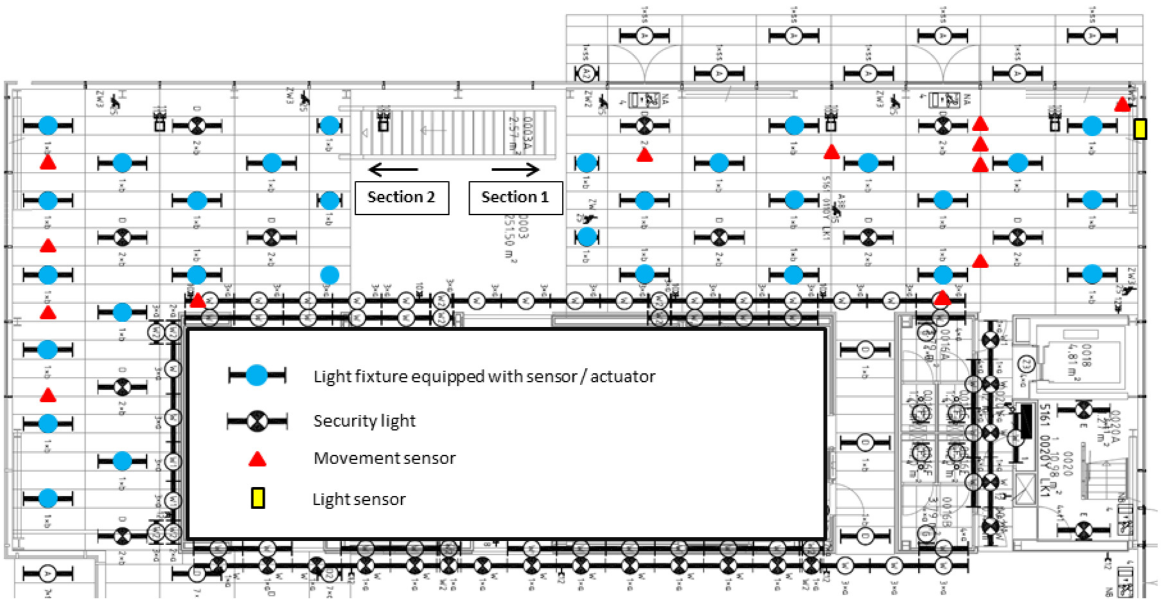


Fig. 6. Schematic representation of the restaurant and deployed devices. The legend only describes the symbols of interest to the experiment.

Table 4
Daily total electricity consumption (kWh) and savings with respect to the use of movement sensors during two weeks of using our system.

Day	Movement sensors	Our system	Savings	Day	Movement sensors	Our system	Savings
Week one				Week two			
Monday	3.63	0.37	89.6%	Monday	3.16	0.73	76.8%
Tuesday	3.94	0.79	79.8%	Tuesday	3.83	1.21	68.3%
Wednesday	3.97	3.97	0%	Wednesday	4.27	0.63	85.3%
Thursday	3.56	1.34	62.2%	Thursday	3.98	0.60	84.8%
Friday	5.19	4.10	21.1%				

6.2.1. *Energy savings*

Observing the measurements gathered in February 2015, when there is manual control in the restaurant, we found that the average time point when the lamps are turned on by the building cleaners is 6:30 a.m; the lamps stay turned on until around 8 p.m., usually switched off by the security personnel. The average consumption per working day in the restaurant is 14 kWh. In weekends, there is no manual control of the lamps, thus always off.

The use of our system results in intelligent coordination of the restaurant with respect to the natural light and presence of people. This means that there are a plenty of possibilities for lamps to be turned off, which provides for energy saving. Fig. 7 shows the daily average electricity consumption when the lamps are controlled manually, when the lamps are triggered by movement sensors, and when our system is used in the restaurant. Fig. 8 shows the intelligent use of lamps and therefore electricity in each day. Fig. 8(a) and (c) illustrate the estimations if only movement sensors would have been used in the first week and second week, respectively. Fig. 8(b) and (d) depict the results of using our system in the first week and second week, respectively. We also include the estimations of consumption if manual control would have been used in all cases.

In addition, the figures include weekends when there is no manual control provided regularly. Though the presence in the restaurant at evenings and during weekends is rare, there are still special occasions that our system encountered without any intervention (see Friday evening and Saturday in Fig. 8(b)). These demonstrate that our system makes the restaurant truly adaptable to the happenings within. To have a fair comparison, we assumed that in cases of special occasions, the lamps in the restaurants would have been manually turned on. Also, one can notice that Friday in the second week is a special occasion too, that is, a holiday. On the other hand, the figures show that our system significantly reduces the consumption of electricity when compared to using movement sensors only (for example, compare Mondays in Fig. 8(a) and (b)).

Finally, Table 4 shows the percentage of savings for each day of the two weeks when compared to using only movement sensors. Wednesday and Friday of the first week have lower savings compared to other days, which is due to the worse weather conditions in these two days. In summary, the average savings of electricity between the scenario of manual control and movement sensors is 71%, the average savings between the scenario of manual control and the one with our system is 89%, and the average savings between the scenario of movement sensors and our system is 61%.

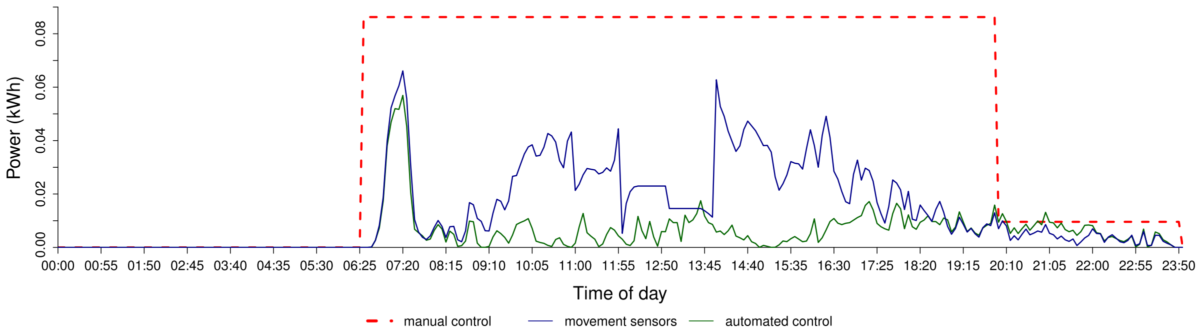


Fig. 7. Comparison of daily average electricity consumption between using manual control, movement sensors and our system to control lamps.

6.2.2. *Economic savings*

We can also have an insight into the amount of money that needs to be paid for the periods of manual control versus our system. Of course, the proportion between the two cases is the same as with the energy consumption, and the price for an average day when our system is used is €0.34, given €0.22 as the kWh tariff.[2] Even in the worst case during working hours, which would happen when weather conditions are worse than on average and the restaurant is visited more than usual, the price resulting from the use of our system stays strictly within the boundaries of the one paid if lamps are manually controlled. Considering a monthly bill paid for electricity when manual control is used, the use of our system implies economic savings such that allow for paying around 7 months in total using the same amount of money as in the bill. That is, while the manual control costs €746 annually, we may pay €88 for electricity annually when our system is used in the restaurant.

6.2.3. *Usability*

We perceive usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" — a usability definition within the ISO 9241 standard for ergonomics of human–computer interaction.[3] Usability testing then provides a way to evaluate such extent of products and systems [39]. To perform the usability testing, that is, a survey, we prepared a questionnaire following the guidelines in [19]. In particular, we identify two groups of users, one experiencing the system during lunch, and another one outside lunchtime. The attitude of occupants towards our system, such as the use of sensors, switching lamps (more often than usual), automation of tasks, etc., defines their acceptability. The need for occupants to understand how to use our system defines learnability. The satisfaction of users with the overall system is related to system effectiveness, and user satisfaction with the time the system takes to perform its tasks relates to system efficiency. While this aspect is more technical, occupants can still evaluate how they perceive reactions of our system. Since our system is unobtrusively integrated into the restaurant, there is no actual use of the system. However, occupants may have different requirements for the system. For example, occupants having lunch may not have the same expectations for the level of light as compared to the ones reading or working in the restaurant outside lunchtime. Finally, we use two Likert scales each with five levels to have an informed way of defining the actual usability of the system [40]. For example, the format of the first scale includes the levels: totally disagree, disagree, neutral, agree and totally agree. Some questions have an additional category that captures the situation when occupants do not have an answer for or cannot answer a respective question. In the end, our questionnaire has 24 questions, two with multiple choices and the rest with the items on the Likert scales.

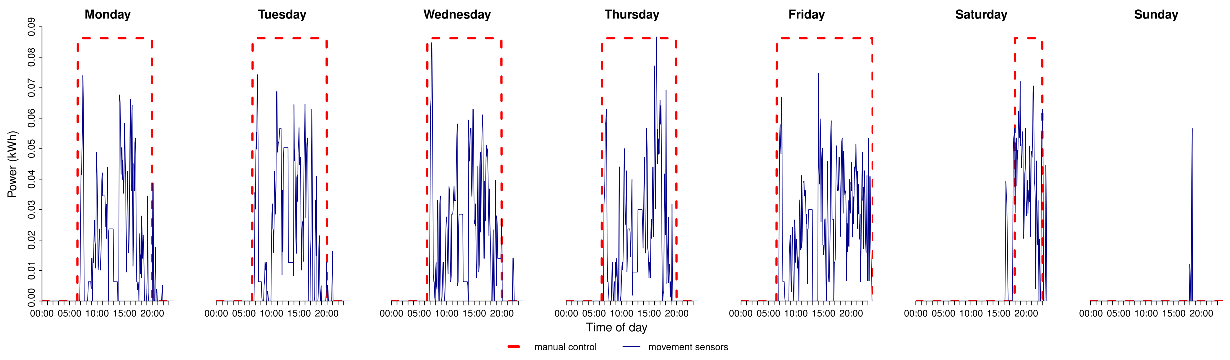
Set-up

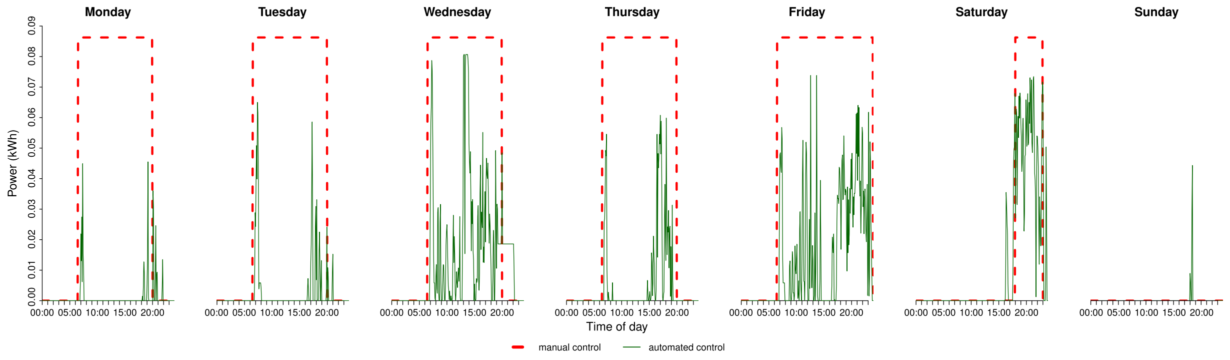
We conducted the survey on the group of occupants experiencing the system during lunch (L), and the one outside lunchtime when working/reading (W). The inputs for the group L are collected on 7 and 9 April of 2015. The total number of inputs is 54. Most of the participants are visitors of the restaurant (57%) while the others are occupants working or studying in the building. Most of the participants are those who use the restaurant only for lunch (96%) while the rest use it both for lunch and studying or working. Participants of this group believe that they are aware of the sustainability issues (83%) and engage in an environmentally friendly behaviour (79%). The majority of participants are familiar with automated control in buildings (64%) and lamps triggered by movement sensors (83%).

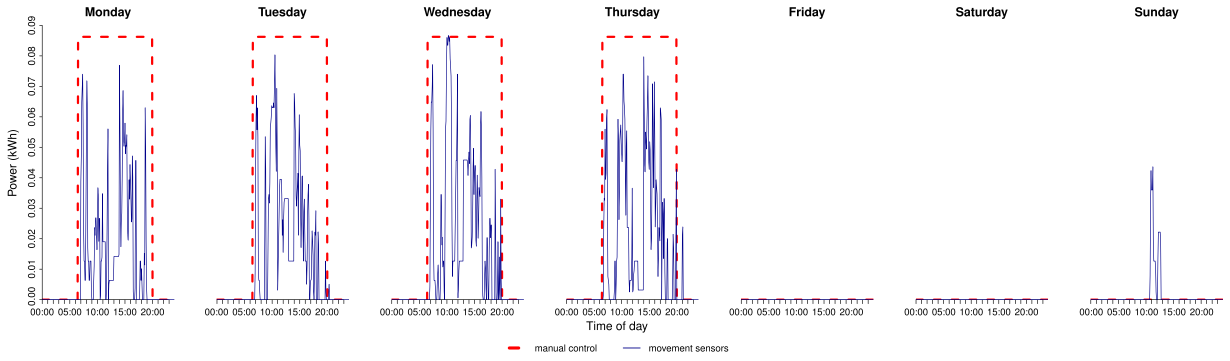
The inputs for the group W are collected on the 4th and 7th May of 2015. The total number of inputs is 18. Most of the participants are students (72%). The group W also consists of participants who believe that they are aware of the sustainability issues (78%) and those that engage in an environmentally friendly behaviour (89%). More than half of the participants are familiar with automated control (55%) and lamps triggered by movement sensors (67%).

2 www.milieucentraal.nl

3 www.en.wikipedia.org/wiki/ISO_9241


(a) Comparison of electricity consumption between using manual control and movement sensors for each day during week one.


(b) Comparison of electricity consumption between using manual control and our system for each day during week one.


(c) Comparison of electricity consumption between using manual control and movement sensors for each day during week two.

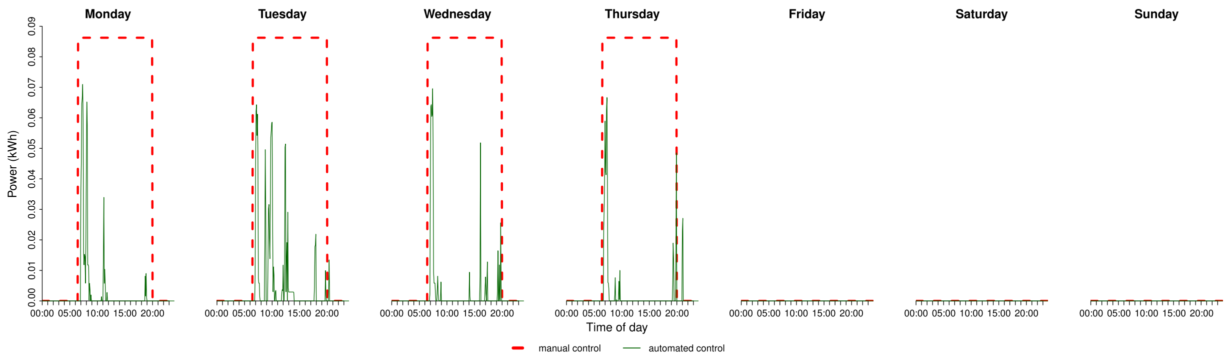

(d) Comparison of electricity consumption between using manual control and our system for each day during week two.

Fig. 8. Comparison of electricity consumption between turning on lamps at fixed times (manual control), using movement sensors, and using our system to control lamps during two weeks.

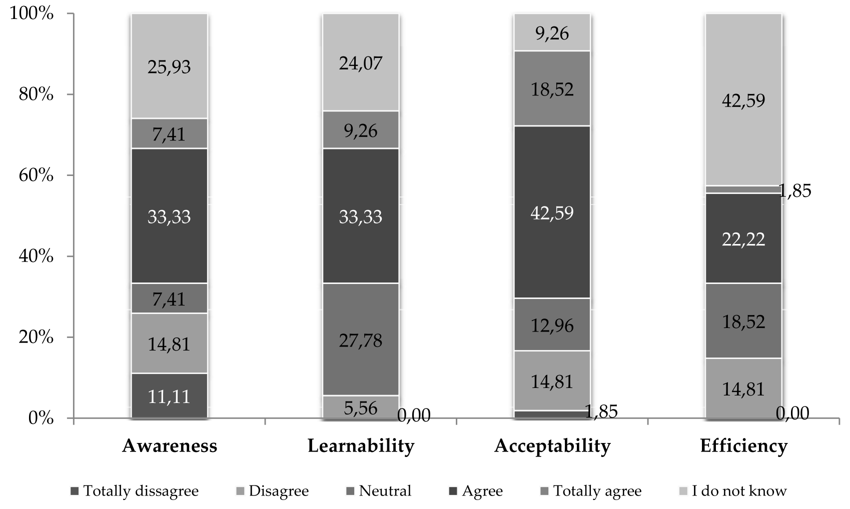


Fig. 9. Results from the occupants of the restaurant during lunch with respect to several aspects.

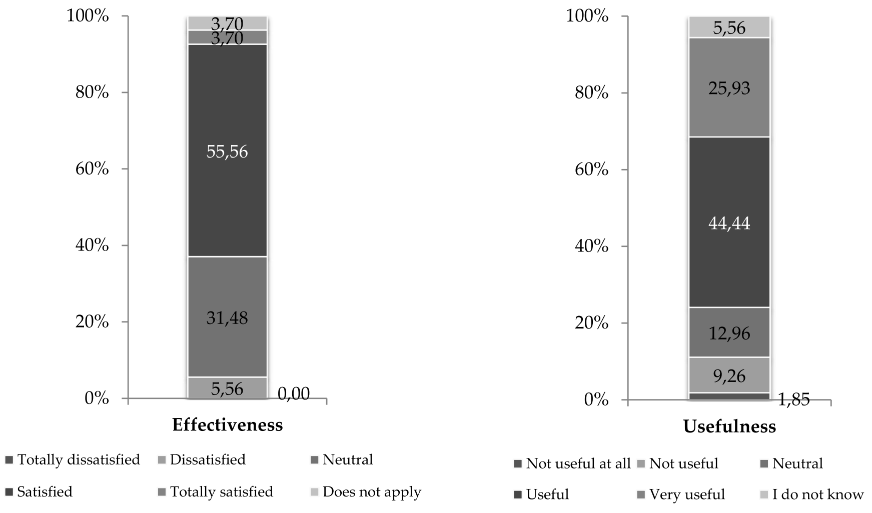


Fig. 10. Results from the occupants of the restaurant during lunch with respect to the effectiveness and usefulness of the system.

Results

We organise the results of the survey on the basis of awareness, perception, acceptability, learnability, efficiency, effectiveness, and usefulness. The results of the group L are shown in Figs. 9 and 10. The group L shows partial awareness with one third of the participants stating that they are aware of the system. The perception of the group L is that our system saves energy (65%), and considers the natural light level (54%) and people's presence (72%). This implies that the majority of this group have good intuition or are well informed. Regarding the learnability, 43% of the group state that it is easy to use the system, while more than half are neutral or do not know the answer to this question. The majority of the participants find the system to be acceptable, while 17% think that the system causes distractions. As for the system efficiency, the majority of the participants do not know if the system reacts immediately to changes or their answer is neutral. When it comes to the effectiveness, 59% of the participants state that they are satisfied with the system, 31% are neutral and only 6% are dissatisfied. Finally, the majority of participants state that they find the system to be useful.

The results of the group W are illustrated in Figs. 11 and 12. The group W also shows moderate awareness with a bit more than one third of the participants stating that they are aware of the system. The perception of the group W is that this system saves energy (78%), and considers people's presence (67%). Half of the participants think that the system also considers the natural light level. This implies that the majority of this group are also well informed regarding energy saving and movement detection. Regarding the learnability, 67% of the group state that it is easy to use the system, while 33% are neutral or do not know the answer to this question. 83% of the participants find system to be acceptable, while none of the participants think that the system causes distractions. As for the efficiency of the system, the majority of the participants do not know if the system reacts immediately to changes or their answer is neutral (72%). When it comes to the effectiveness, 72% of the

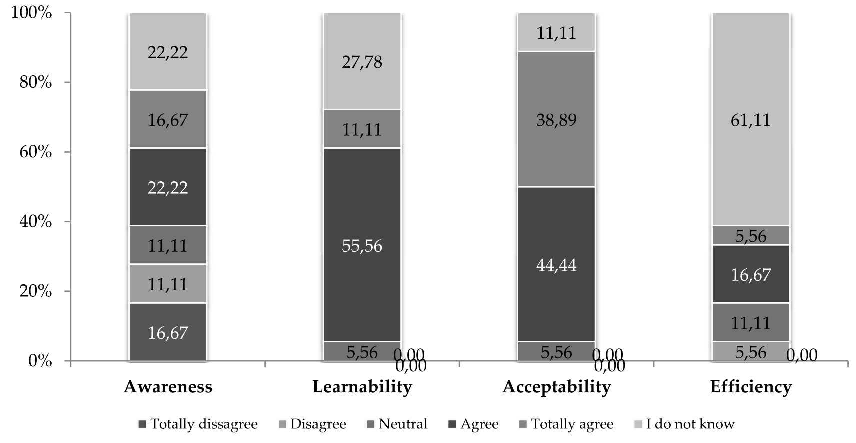


Fig. 11. Results from the occupants of the restaurant outside lunchtime with respect to several aspects.

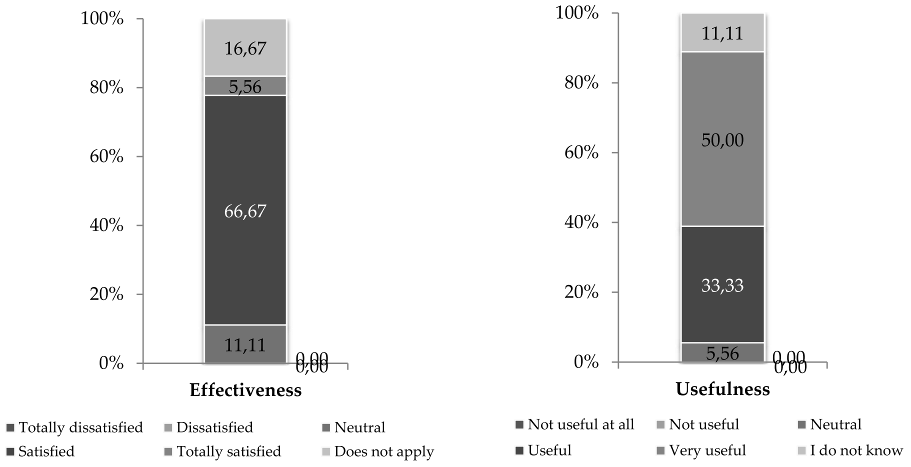


Fig. 12. Results from the occupants of the restaurant outside lunchtime with respect to the effectiveness and usefulness of the system.

participants state that they are satisfied or very satisfied with the system, 11% are neutral and none of the participants are dissatisfied. Finally, the majority of the participants (83%) state that they find the system to be useful.

6.2.4. System performance

We are interested in the scalability of the system. The system is scalable if it is capable to cope and satisfactory perform under varying sizes of building coordination problems. Given that planning is computationally expensive, the system performance depends on the capabilities of the planning technique to address a varying size of HTN planning problems.

During the run of our system in the restaurant, we recorded the HTN planning problem generated on each invocation of the planner. The smallest planning problem consists of 177 state elements, while the biggest one of 207. The number of tasks in the initial task network is constant and equal to 13 (one task for each area). Then, the size of HTN planning problems encountered during the run varies between the ones of the smallest and biggest planning problem. Furthermore, the average number of HTN planning problems solved is 264, the average number per working day is 360, and the average number per weekends/holidays excluding exceptional situations is 31. In fact, two days are exceptions. One working day that has almost twice more invocations of **SH** than the average number of invocations in working days. The second is a weekend day, which has a number of invocations unusual for weekends.

Given the number of HTN planning problems that need to be solved per day, and that planning is a computationally expensive task, we want to see whether the planner can remain practically useful in building environments larger and more complex than the restaurant. We have therefore executed a set of performance tests on a Intel Core i7-3517U @1.90 GHz, 8GB RAM machine running Windows 8.1 and Java 1.8.0_31, a machine different than the deployment one. The tests are based on the HPDL problem description created during the system run in the restaurant, and use the same domain model. We run **SH** on each HTN planning problem three times, and measure and present mean values.

We generated two sets of HTN planning problems by changing their load profiles. In the first set of tests, we evaluate the performance in terms of scalability of the number of tasks in the initial task network under a constant number of lamps

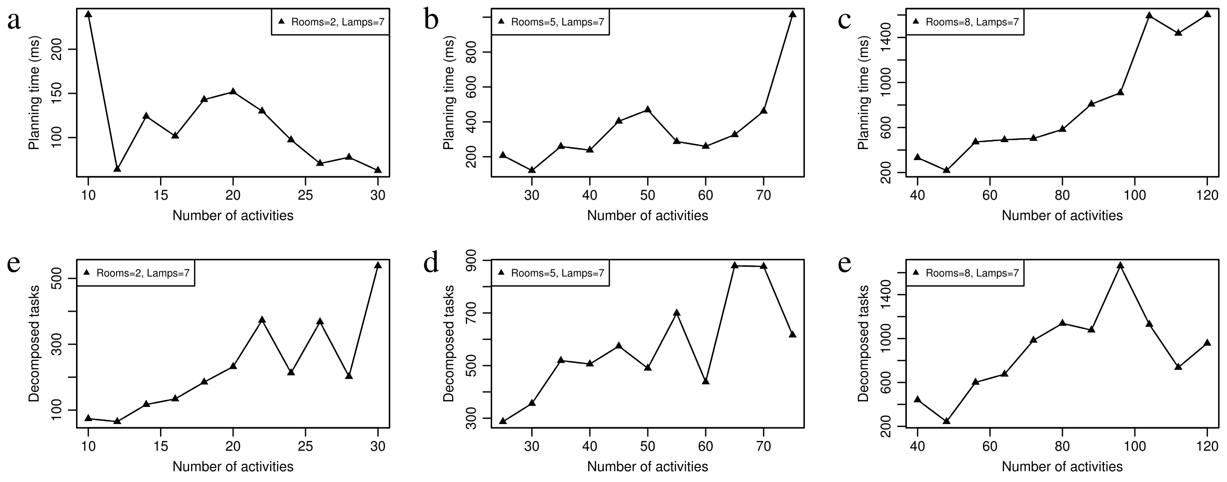


Fig. 13. Performance results of **SH** when scaling the number of tasks in the initial task network (the number of lamps per area).

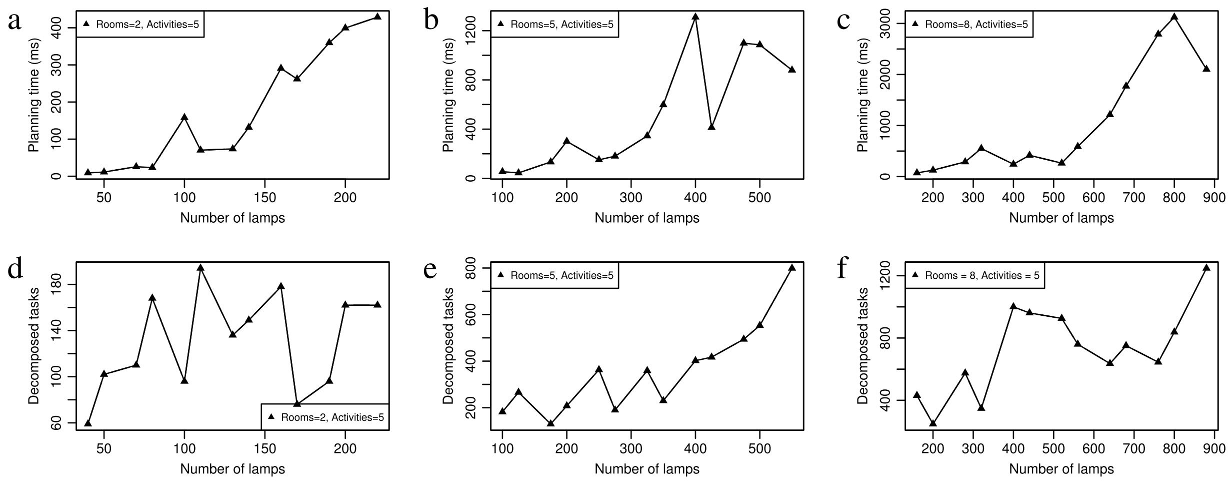


Fig. 14. Performance results of **SH** when scaling the number of lamps per area.

per area. We are interested in the behaviour of **SH** when the size of the restaurant increases. Fig. 13 shows the scaling of **SH**. While the number of lamps per area is fixed, the charts in Fig. 13(a), (b) and (c) indicate the planning time and the charts in Fig. 13(d), (e) and (f) depict the number of tasks decomposed both in function of the number of tasks in the initial task network. In addition, moving from left to right charts, we increase the number of rooms too. Looking at Fig. 13(a), it is difficult to assess the behaviour of the planning system. From Fig. 13(b) and (c), one can notice more or less a gradual rise of the planning time up until 70 simultaneous activities in Fig. 13(b), and until 100 simultaneous activities in Fig. 13(c) where the number of rooms is higher. After these points, the planning time increases steeply where the worst case is just above 1.5 s. Fig. 13(d), (e) and (f) demonstrate that the number of decomposed tasks increases on a regular basis, though one can still notice zigzag curves.

In the second set of tests, we evaluate the performance in terms of scalability of the number of lamps when the number of areas (and therefore simultaneous activities) is constant. Considering that lamps can be exclusively in one area or shared with other areas through the `near-by` predicate, this increases the difficulty of solving a planning problem. Thus, the number of predicates related to location properties increases too. Fig. 14 depicts the results. The organisation of the charts is the same as in Fig. 13. With respect to the planning time, one can notice that the number of lamps and their relationships with the environment layout affect the performance of **SH**. When the number of lamps is 200, the planning time is almost the same in all three charts (Fig. 14(a), (b) and (c)) despite the fact that the number of rooms across which the decision is made is increased (from 2 through 5 to 8). The curves are gradually rising, and **SH** needs around 3 s to deal with 800 lamps.

In comparison with Figs. 13(d), (e) and (f), 14(d), (e) and (f) do not present easily identifiable patterns. Fig. 14(e) depicts an oscillating curve up until 400 lamps, when the number of decomposed tasks increases fast. In the case of Fig. 14(f), one extreme is at 400 lamps, when the number of decomposed tasks drops gradually, and then a rising curve to the other extreme for 900 lamps.

7. Discussion

The first case of evaluation is an operational one where the outcome of activity recognition is gathered from a completely real setting, while the controlling part is simulated. Consequently, two remarks are necessary: the sensor information about the natural light level is constructed from the information on weather conditions of the three days of experiments; and there is no actual execution of plan actions.

The high percentage of savings with respect to the use of lamps is due to the three sunny days, resulting in lamps being turned off most of the time. The potential savings from the use of workstations are due to the control of computers and monitors according to the activities that office occupant perform. It is a known issue that energy waste is caused by leaving computers on, e.g., [34–37]. It appears that office workers leave their computers turned on when they are not being used, especially overnight and during the weekends. For example, of all computers in offices, 64% are powered on even after-hours according to [34], 50% in US and 44% in Europe are never or rarely turned off as reported in [36], and 50% are still running during off-peak hours as found out in [37]. If a computer is turned on, it does not necessarily mean that the office worker is working continuously during the whole working time. In fact, computers are turned on and left unattended for about 28% of the time [35]. Importantly, only 6% of all computers that are not turned off made use of the available power management options, such as sleep mode [34]. Regarding the monitors attached to the computers, 82% of all monitors are not turned off, 75% of all monitors made use of their power management options, while 21% of the monitors are always powered on [34]. According to [35], 31% of office workers assume that their computer has power management options enabled, but, in reality, it is often the case that such options are not active. In fact, the sleep mode is disabled by default at our own university. While our assumption to have workstations and lamps turned on continuously during working hours may seem hypothetical at first, it actually reflects the reality and may even lead to an underestimation of the potential energy consumption when regular control is used.

Since the actual deployment and use of our system was in the case of the restaurant, we provide a more extensive discussion for this case. In fact, during the first three weeks of monitoring in the restaurant, we encountered several technical issues, e.g., the sensor network stopping to respond or responding irregularly. These issues did not affect negatively the results as we were collecting data only for the baseline. The actions we took to address the issues are part of the building life-cycle, and thus resulted in a longer monitoring period of the restaurant. On the other hand, a small amount of the data collected from the movement sensors during the two weeks appeared to be incompletely stored in the repository. The incompleteness refers to missing time points when movement sensors report true value. While this is an issue that should be considered to improve the reliability of the system, it does not negatively affect the results on electricity savings achieved with our system because the missing points would only increase the consumption in the case of using only movement sensors.

A timeout was applied to the durability of the presence activities with the goal to prevent triggering of new events at changes of variable values that are expected at specific time periods. For example, during lunchtime when the frequency of people in the restaurant is drastically higher than the rest of the day, we can avoid triggering unnecessary events by the movement sensors for 30 min. At the same time, the timeout is used to reduce the occupant annoyance with the need of intentionally produced movements when the detectors are unable to sense their motion. We set three different timeouts manually for the presence activity for four different periods of a day. Based on our results, we conclude that the timeout approach and timeouts themselves are robust enough to be applied for other types of rooms for the presence type of activity. Another option is to adjust timeouts dynamically and automatically from observations over a certain period of the building operation. Additionally, not only the data coming from sensors but also the feedback and preferences from occupants can be considered to learn timeouts over time.

One of the strongest motivators to migrate towards intelligent building management systems is energy saving. Our system offers energy savings, which are due to the coordination of lamps given weather conditions, presence of people, and a set of minimum requirements for satisfactory level of light. Looking at the system itself, it consists of 30 Plugwise devices, 14 sensors, one thin client, and one server that consume 3.3 W, 6.7 W, 4 W, and 365 W, respectively. The amount of energy consumed by the actuators and sensors thus represents a negligible fraction of the total consumption. While the thin client is additional hardware we had to consider, the server represents a computer already in use, which adds an insignificant amount to its current consumption. This situation may be negatively affected in cases of network connection drop outs and when the system needs to use the sensors and actuators at the time of the drop-out. Until these become available again, the system will continuously try to adjust the restaurant, resulting in a use of the processing power of the server more than usually. As for the thin client, its daily consumption is 0.8% from the savings obtained from an average day of energy consumption.

Even if we disregard the costs of software development, the necessary hardware and its deployment has a non-negligible cost. In the literature, however, there is no tool for estimating yearly savings (due to an adoption of a specific building management system), and consequently the number of years needed to return the investment [2]. Nevertheless, we provide the following insights for our solution. Two situations, which do not occur in the case of manual control but happen when using our system (and when using movement sensors only), may imply leakage of money to be encountered during evenings in working days and on weekends. Assuming that building conditions require a higher level of light (at night), the situations involve turning on lamps due to people passing through and not staying in the restaurant. It is difficult to extrapolate such situations from the data, but one obvious example is Sunday in week one in Fig. 13. The price paid for this situation is negligible, or 0.78% from the amount of money we save in an average day. Hence, the benefit of using our system is reflected in an annual savings of €658. If we consider the equipment that needs to be bought and deployed, the costs for 30 Plugwise devices, 14 sensors, one thin client, and one server are €990, €2040, €200, and €2700, respectively. Then, the payback period for the investment in our system would be nine years. Nevertheless, we expect that the cost of hardware will fall in the future, making the system even more attractive.

Figs. 13 and 14 show that **SH** can successfully compute plans in just few milliseconds for large planning problems representing real-world situations. The experiments in which **SH** takes up to 3 s to find a solution are much larger but unrealistic and to the extreme. In reality, there will be more than one instance of **SH** to cover such a space. Nevertheless, one instance of **SH** takes, for example, only 3 s to find out which lamps to turn on or off from a pool of 800 lamps. This is a benefit rather than a limitation. With these performance results, we show that our system is a potential solution for large and complex open-space building environments.

The performance of **SH** is closely related to the quality of the domain model. In our case, the quality depends on the capabilities and expertise in pervasive computing of the domain author. There are still no ways of providing objective measures of the quality of the domain model, and the relative effort needed to formulate such model [21]. On the other hand, learning the domain model can prove to be more appropriate approach. There is a large body of literature dealing with acquisition of HTN domain knowledge by using learning and semi-learning approaches, e.g., [41–46]. These studies can be exploited to further investigate the benefits of learning planning domain models in pervasive computing.

In addition to the performance evaluation, we conducted a usability study. The study provided insights into the attitude and satisfaction of the restaurant occupants with the usefulness and effectiveness of our system. Beside the realistic nature of the setting, the participants are not a random sample of the building population. The participants have specific social and academic characteristics that are correlated to the university environment. Particularly, the participants are well informed about sustainability issues and behaviour. In addition, the participants are familiar with automated control and sensors in buildings. These factors prevent us from generalising to all building population. Moreover, even though the participants were exposed to a mandatory 'use' of our system (as opposed to volunteers), from the results it can be argued that these participants are highly motivated to engage in environmentally friendly behaviour and will therefore be more inclined to accept our system than a general building population. We thus have a second reason to be careful about generalising to the building population.

The use of surveys has well-understood limitations. As the participation in the survey itself is voluntary, the outcome that a survey produces tend to over-represent participants who are most active and motivated [47]. So our results may be most representative of the most active and motivated occupants, which is not a negative thing per se. That is, users often drive innovation, and studying the attitude of unusually motivated users (or lead users) can be a valuable way of "uncovering innovation and identifying barriers for adoption" [48,47]. While we should be careful of generalising to the building population, we should be aware that if current participants are dissatisfied or have difficulties with our system, it is likely that those who follow them will have as well.

Generally, we envision that the system could be easily extended for other office buildings and applied to other types of buildings. For example, if the system is used in care centres, a wide range of activities of both patients and care givers can be anticipated under the protocols of the centres.

Building environments are related to time, inherently uncertain about the current state, and unpredictable at execution time. Our solution can be extended to support temporal associations either during the planning process or as a post planning phase. Instead of being only sequences of services, the plans would be associated with higher execution flexibility (e.g., services executed in parallel). Uncertainty makes building coordination problems more complex, thus harder to be solved. With our orchestration model and strategy, we address only simple issues that may appear during building coordination, such as unpredicted events and service failures. The conditions when and how these issues appear and are solved could be extended and refined in the manner of the orchestration approach proposed in [17].

8. Related work

Our solution is based on service coordination that composes services to anticipate occupant activities and provide new capabilities to building management systems. We therefore look at related work at the intersection of building management systems, service composition for pervasive computing, context modelling and activity recognition techniques.

8.1. Building management systems

Various systems and techniques have been proposed for intelligent building management [2]. A wide variety of these systems have the goal of reducing energy consumption and satisfying user requirements at the same time. In the context of lighting control, there are studies reporting a potential of energy saving up to 58% [49]. We show that with our system, buildings have the potential to save in the order of 80%.

Among the open challenges identified in [2] is the use of planning for automating the management of buildings. Our work makes the first step to face this challenge, and thus is the first attempt to actually bring AI planning in combination with activity recognition into the perspective of building management systems.

8.2. Service composition for pervasive computing

The description, construction and execution of service compositions have been topics attracting much research attention. To provide means for realisation of efficient service compositions, numerous studies focused on service discovery and matchmaking, protocol evolution, QoS awareness, and other challenges.

The construction of service compositions has been mainly accomplished within the scope of workflow management and AI planning. In workflow management, there are static and dynamic workflow generation methods [50]. In the static composition method, an abstract process model is predefined and only the selection and service binding is performed automatically, e.g., [51]. In the dynamic composition one, the process model and service selection are done automatically. In this context, model checking is used for service composition to verify composition correctness, e.g., [52]. More flexible and context-aware approaches for Web service composition have been proposed based on AI planning. While service composition for pervasive computing has a close resemblance with Web service composition, there are two notable differences. First, Web services are distributed on the Internet, thus publicly available, and assumed to be registered to some repository. Pervasive services, in contrast, are commonly part of well-controlled environments. The second and important issue with Web services lies in the lack of consistent semantic annotations such that make Web service composition feasible in practice. Even though various ways to describe Web services exist (e.g., SOAP, WSDL, OWL-S), some already deprecated or never used in practice, the reality of Web services is that they are associated only with syntactic specifications and free-text descriptions, leading to the consideration of Web services as nothing more than data sources [53]. Being part of controlled environments, pervasive services have different characteristics: they tend to be more structured and described using consistent (in-house) ontologies. Corporations are interested in making use of well-established standards and best practices they gain in the domain of service-oriented architectures to support a standardised way of access to services. In contrast to Web service composition, these considerations foreground the possibility to make the composition of pervasive computing applications feasible in practice.

In the area of HTN planning, several studies deal with the composition of Web services most of which focus on services described in OWL-S [21]. OWL-S is a language specifically designed to support the discovery, composition and monitoring of Semantic Web services. In reality, however, the language supports essentially only behavioural descriptions of services [54,53]. Such descriptions seem insufficient to be correctly translated into HTNs, and moreover, inappropriate to reason over. This drawback prevents OWL-S from being used in practical and real-world cases at all. On the other hand, our approach is not dependent on a specific modelling language, but on a formal model that captures the semantics of building environments. Additionally, the studies assume the existence of OWL-S compound Web services which can be translated to HTN methods and compound tasks (for details, see [13]). On the other hand, we do not use any compound services, but we use compound tasks to model the rich knowledge of (or complex processes happening in) building environments.

The use of AI planning in pervasive computing has been a subject of research in many existing studies [19]. The studies usually describe pervasive computing environments intuitively through scenarios and examples where devices provide some operations or services and requests are issued by people or software components. In some cases, a spatial organisation of environments is portrayed too. The basic correspondence that these studies establish between pervasive computing environments and AI planning is by relating services to actions and requests to goals. All other environment-specific knowledge, such as the spatial organisation, is integrated somehow into the planning domain model. Kaldeli et al. [8] take a more transparent approach and provide a formalised transition between their home domain and the planning domain model. On the other hand, we define the building coordination problem and establish a correct correspondence between that problem and an HTN planning problem.

Similarly, one notices the way existing studies deal with uncertainty. Several studies provide intuitive descriptions of the process of execution and monitoring [19]. While we resort to a pragmatic approach to handle inconsistencies, we define the orchestration problem and circumstances under which it can be addressed based on the model proposed in [17]. This enables us to make the first relation of HTN planning to execution semantics in pervasive computing.

Finally, only few studies demonstrate the feasibility of the proposed approaches, and even fewer use planning in actual pervasive computing environments [19]. We employ HTN planning in the Bernoulliborg building at the University of Groningen, and show the feasibility of our solution by quantitative, qualitative, and usability evaluations.

8.3. Context and activity modelling and recognition

We identify occupant activities using activity recognition based on ontological modelling and reasoning. Thus, the context and occupant activities are modelled using ontologies. In addition to the ontology-based approaches, two other prominent ways for context and activity modelling are proposed in pervasive computing, namely object-role based approaches, e.g., [55], and spatial models, e.g., [56]. Each way has its own advantages and disadvantages, and none of them satisfies all the requirements set for context modelling of building environments [57]. We realise an ontology-based approach that models spatial context and occupant activity as it fulfils most of the requirements for pervasive computing applications [11].

In a broader sense, activity recognition based on logical modelling and reasoning overcome some of the disadvantages of the alternative approaches to activity recognition based on symbolic learning, such as the demand of a sufficiently large amount of labelled data needed for training. Instead, without any training data, it is still possible to obtain potential performed activities of occupants in a given context by exploiting logical reasoning [58]. In addition, in the context of building environments, the number of common activities is limited and often well known. Instead of being learned, the activities can be thus predefined by domain and knowledge engineering experts.

In pervasive computing, ontology-based approaches have been used to build activity ontologies, and to recognise activities based on environmental data, e.g., [58,59]. These proposals are however mainly concerned with the support of people monitored for medical or security reasons in residential and household environments. On the other hand, we investigate a solution based on activity recognition with ontological reasoning to support building environments.

Finally, it is theoretically possible to extend and embed other existing context models into the building coordination problem. For example, the context space theory [60] covers many aspects of the context we deal with. However, it does not define occupant activities, which are a crucial element and one of the linking aspects for HTN planning problems.

9. Conclusions

We solve building coordination problems intelligently and automatically by combining AI planning and activity recognition for the first time. We showed how a building coordination problem that anticipates occupant activities is defined, and how a correct correspondence between this problem and an HTN planning problem is established. This correspondence is an improvement over existing approaches on AI planning for pervasive computing. It allows for the use of plans computed for the HTN planning problem as sound solutions to the building coordination problem. Moreover, we considered orchestration with defined semantics for plan execution under uncertainty. The solution is therefore able to fully support the capabilities of building management systems: from sensing and recognising occupant activities, planning for composing services, to executing services upon devices.

The developed system prototype is used in the offices, the social corner and the restaurant of our own building at the University of Groningen. We showed that the combination of HTN planning and activity recognition can bring savings that go beyond the estimations associated with the lighting control of existing automated solutions. The return on investment for the adoption of our solution in the restaurant is nine years. We showed that the performance of **SH** in larger and open-space environments can stay within few seconds. The possibility of the system to support different occupant activities and to be applied in diverse types of rooms led us to conclude that the system can be deployed to the whole building. While we could not generalise the results of our usability study to the whole building population, we can conclude that active and environmentally motivated occupants are satisfied with the system and ready to accept intelligent systems as ours.

Acknowledgements

We thank Ellen van der Werff for the feedback on the questionnaire. We are grateful to the participants in the usability study, and to all occupants of the Bernoulliborg restaurant for the patience during the period of the system deployment. We thank Pieter Noordhuis for the initial implementation of the **SH** planner. The work is supported by the Dutch National Research Council (NWO) Beijing Groningen Smart Energy Cities (BIGS) Project, contract no. 467-14-037.

Appendix A. Background

A.1. Activity recognition

We use an ontology-based technique for activity recognition that applies ontological modelling and reasoning. We go beyond simple modelling of activities, sensors, devices and locations in a taxonomical way as we also express semantic relationships and constraints between these ontologies. By having a complete picture of a given context, logical reasoning can be performed, making it feasible to recognise the possible activities of the occupants.

A.1.1. Ontological modelling of occupant activities

The set of all possible activities Act, as defined in Definition 2, occurring at each and every location is represented by an ontology of activities named ***Activity***, while the set of location L is represented by an ontology of locations called ***SpatialContext***. At the same time, an ontology represents the set of building variables V called ***ArtefactualContext***. Moreover, we model the set of semantic relationships and constraints between the three ontologies. In our ontologies, each location $l \in L$ is an entity in the ***SpatialContext*** ontology while the artefacts, i.e., sensors and actuators, at each location l are indicated in the ***ArtefactualContext*** through the relation *hasArtefact*. Each activity act in ***Activity*** *happensIn* a specified location $l \in L$ and *hasDetected* the involvement of some artefacts in ***ArtefactualContext.*** Detailed analysis and design of the ontologies are discussed in [10].

A.1.2. Ontology-based activity recognition

The activity recognition function ar, as given in Definition 3, is represented as a process of ontological reasoning over the defined ontologies using the ontological relationships between the ontologies. Algorithm 2 provides a high-level overview of the ontological reasoning process. The algorithm works with an infinite cycle in which it reads all the pushed sensor data from the environment variables V at the end of each time interval and inputs the data to an ontological reasoner, e.g., the *HermiT* inference engine (www.hermit-reasoner.com). The reasoner also takes the terminological part of the ontologies to derive the specific activities performed at every activity area in the current time interval.

Algorithm 2 Activity Recognition

```
Require: $V$: Environment variables $V = v_i$; ontologies with their instances and the interrelationships between them.
Ensure: $Act$: Activities performed at all locations $l_i \in L$
1: $read(V)$
2: Assign each $v_i \in V$ to the corresponding instance in the ArtefactualContext ontology.
3: if <Run for the first time> then
4:     for <each of the instances> do
5:         Retrieve all locations associated via the relationship liesIn.
6:         Retrieve associated activities via the relationship hasDetected.
7:     end for
8:     for <each of the retrieved locations> do
9:         Retrieve all activities from the retrieved activities via the relationship happensIn.
10:        Assign the occurring activities $\{act\}$
11:        Update $Act$
12:    end for
13: else
14:    if <a change of $v_i \in V$ is received> then
15:        Retrieve all locations associated via the relationship liesIn.
16:        Retrieve associated activities via the relationship hasDetected.
17:        Retrieve all activities from the retrieved activities via the relationship happensIn.
18:        Assign the occurring activities $\{act\}$
19:        Update $Act$
20: return $Act$
```

At the initial state, the algorithm takes as input $E = \langle V, L\rangle$. The values of building variables V represent the current states of all instances in the ontology ***ArtefactualContext*** together with the terminological part of the other ontologies, that are (1) ***Activity*** for the set of all possible activities $\{oa\}$ of the environment, (2) ***SpatialContext*** for the set of all location L. The algorithm performs the ontological reasoning process in order to initialise the output, that is, the set of occurring activities Act at corresponding locations in L. Whenever there is a change in the value of some $v_i \in V$, the algorithm updates the corresponding instance in the ontology ***ArtefactualContext***. The algorithm performs an ontological query process in order to determine the activities $\{oa\}$ associated with the changed v_i. After that, the algorithm updates the set of occurring activities Act.

Theorem 3. *Let E be a building environment. If activity recognition always follows the change of values in V at all locations in L, then activity recognition is complete.*

Proof. Let us assume that the algorithm is informed about the changes of the values of building variables at all building locations. For each change of a variable value, the algorithm updates the corresponding instance in the ontology ***ArtefactualContext***, and triggers reasoning over the ontologies. Assuming that the reasoning over ontologies always identifies activities, if any, then the algorithm is complete. □

A.2. HTN planning

We use an extension of the formalism for state-based HTN planning [21] to support numerical state variables, which are useful constructs for pervasive computing environments [8]. What follows defines in essence the notations of the Hierarchical Planning Definition Language (HPDL) [26,27], which partially follows the *de facto* standard PDDL2.1 framework for classical planning [61].

All sets in the following definitions are finite and non-empty if not stated otherwise. A primitive name is an expression of the form $pn(\tau)$, where pn is a primitive symbol, and $\tau = \tau_1, \ldots, \tau_n$ are terms. A compound name is defined similarly. We refer to the set of primitive and compound names is as a set of task names TN.

Definition 7 (*Domain Theory*). A *domain theory* $\mathcal{D}$ is a tuple $\langle \mathcal{Q}, \mathcal{V}, \mathcal{T}\rangle$, where:

- $\mathcal{Q}$ is a set of predicates.
- $\mathcal{V} = \{\bar{v}_1, \ldots, \bar{v}_n\}$ is a set of variables. Each $\bar{v}_i \in \mathcal{V}$ ranges over a finite domain D_i.
- $\mathcal{T}$ is a set of tasks. The set of tasks $\mathcal{T}$ consists of a set of primitive and a set of compound tasks, where:

Table B.5
Table of notation.

Section 2		
d	$\triangleq$	Device
s_d	$\triangleq$	Sensing service of device d that returns an output
V	$\triangleq$	Set of building variables
D^v	$\triangleq$	Domain of values for a variable $v \in V$
L	$\triangleq$	Set of locations
E	$\triangleq$	$\langle V, L \rangle$ is a building environment
$Prop$	$\triangleq$	Set of building properties
Act	$\triangleq$	Set of occupant activities
ar	$\triangleq$	Activity recognition function
Ψ	$\triangleq$	Set of building conditions
P^B	$\triangleq$	$\langle E, \Sigma, Prop, Act \rangle$ is a *building coordination problem*
a_d	$\triangleq$	Acting service of device d that changes a value
A	$\triangleq$	Set of acting services
α	$\triangleq$	Sequence of acting services that is a satisfying coordination for P^B
Section 4		
$\mathcal{S}$	$\triangleq$	Set of states
$\mathcal{E}$	$\triangleq$	Set of events
γ	$\triangleq$	$\mathcal{S} \times A \times \mathcal{E} \rightarrow \mathcal{P}(\mathcal{S})$ is a state transition function
Σ	$\triangleq$	$\langle \mathcal{S}, A, \mathcal{E}, \gamma \rangle$ is an orchestration model
P^O	$\triangleq$	$\langle \Sigma, V, Act \rangle$ is an *orchestration problem*
Π	$\triangleq$	Orchestration plan
Π'	$\triangleq$	Reduced orchestration plan
Appendix A.2		
pn	$\triangleq$	Primitive symbol
cn	$\triangleq$	Compound symbol
τ	$\triangleq$	$\tau_1, \ldots \tau_n$ are terms
$pn(\tau)$	$\triangleq$	Primitive name
$cn(\tau)$	$\triangleq$	Compound name
TN	$\triangleq$	Set of primitive and compound names (task names)
$\mathcal{Q}$	$\triangleq$	Set of predicates
$\mathcal{V}$	$\triangleq$	Set of (planning) variables
D_i	$\triangleq$	Domain of values for a variable $\bar{v}_i \in \mathcal{V}$
o	$\triangleq$	Primitive task or operator
$pre(o)$	$\triangleq$	Preconditions of o
$eff(o)$	$\triangleq$	Effects of o
t	$\triangleq$	Compound task
M_t	$\triangleq$	Set of methods
tn	$\triangleq$	Task network
T_n	$\triangleq$	$\subseteq TN$
$\mathcal{T}$	$\triangleq$	Set of primitive and compound tasks
$\mathcal{D}$	$\triangleq$	$\langle \mathcal{Q}, \mathcal{V}, \mathcal{T} \rangle$ is a domain theory
s	$\triangleq$	State
$s[o]$	$\triangleq$	New state by applying o to s
$\mathcal{P}$	$\triangleq$	$\langle \mathcal{D}, s_0, tn_0 \rangle$ is an *HTN planning problem*
π	$\triangleq$	Sequence of primitive tasks or plan

- A primitive task (also an operator) o is a tuple $\langle pn(o), pre(o), eff(o) \rangle$, where:
  - $pn(o)$ is a primitive name,
  - $pre(o)$ are preconditions. A *condition* is a pair $\langle p(cond), nc(cond) \rangle$, where $p(cond) \subseteq \mathcal{Q}$ is a set of predicates and $nc(cond)$ is a set of numerical constraints. A *numerical constraint nc* is a tuple $\langle exp, \circ, exp' \rangle$, where exp and exp' are expressions, and $\circ \in \{\langle, \leq, \geq, \rangle\}$ is a relational operator. An *expression exp* is an arithmetical expression constructed by using a binary operator $\diamond \in \{+, -, *, /\}$ over $\mathcal{V}$ and $\mathbb{Q}$.
  - $eff(o)$ are effects. An *effect* is a pair $\langle p(eff), nc(eff) \rangle$, where $p(eff) \subseteq \mathcal{Q}$ is a set of predicates, and $nc(eff)$ is a set of numerical effects such that for all $\langle \bar{v}_i, assign, exp \rangle, \langle \bar{v}_j, assign, exp' \rangle \in nc(eff)$ it holds $i \neq j$. A *numerical effect* is a tuple $\langle \bar{v}_i, assign, exp \rangle$, where $\bar{v}_i \in \mathcal{V}$, $assign \in \{:=, +=, -=, *=, /=\}$ is an assignment operator, and exp is an expression.
- A *compound task t* is a pair $\langle cn(t), M_t \rangle$, where $cn(t)$ is a compound name, and M_t is a set of methods. A *method m* is a pair $\langle pre(m), tn(m) \rangle$, where $pre(m)$ are preconditions and $tn(m)$ is a task network. A *task network tn* is a pair $\langle T_n, \prec \rangle$, where $T_n \subseteq TN$ are task names, and $\prec$ is the order of task names in T_n.

A state s is a pair $\langle p(s), v(s) \rangle$, where $p(s) \subseteq \mathcal{Q}$ is a set of predicates and $v(s) = (v_1(s), \ldots, v_n(s)) \in \mathbb{Q}^n$ is a vector of rational numbers. Basically, $p(s)$ represent predicates that evaluate to true, and each $v_i(s)$ is the value of $\bar{v}_i$. We say that a numerical constraint is *satisfied* in state s if the value of exp in s is in relation $\circ$ to the value of exp' in s. A numerical effect is *applied* in a state s by updating the value of $\bar{v}_i$ in s with the value of exp in s using the assignment operator $assign$. A primitive task o is *applicable* in a state s if $s \models pre(o)$. $s \models pre(o)$ holds iff $p(pre(o)) \subseteq p(s)$ and all numerical constraints in $nc(pre(o))$ are satisfied in s. Applying o to s results into a new state $s[o] = s \cup eff(o)$, where $p(s[o]) = p(s) \cup p(eff(o))$ and $v(s[o])$ is the vector of values generated by applying all numerical effects in $nc(eff(o))$ (unaffected values are left unchanged). A method m is *applicable* in a state s if $s \models pre(m)$. Akin to a primitive task, $s \models pre(m)$ holds iff $p(pre(m)) \subseteq p(s)$ and all numerical constraints in $nc(pre(m))$ are satisfied in s. Given a compound task t and a method m such that $m \in M_t$, applying m to s results into a task network $tn(m) = (s[m], t)$.

Having the definition of the domain theory and the semantics, we can define the HTN planning problem and its solution.

Definition 8 (*HTN Planning Problem*)**.** An *HTN planning problem* $\mathcal{P}$ is a tuple $\langle \mathcal{D}, s_0, tn_0 \rangle$, where $\mathcal{D}$ is the domain theory, s_0 is an initial state, and tn_0 is an initial task network.

A sequence of primitive tasks $o_1, \ldots, o_n$ is *applicable* in s if there are states $s_0, \ldots, s_n$ such that $s_0 = s$ and o_i is applicable in s_{i-1} and $s_{i-1}[o_i] = s_i$ for all $1 \leq i \leq n$.

Definition 9 (*Solution*)**.** Given an HTN planning problem $\mathcal{P}$, a *solution* to $\mathcal{P}$ is a sequence of primitive tasks $o_1, \ldots, o_n$ applicable in s_0 by decomposing tn_0.

We refer to the solution of an HTN planning problem as plan π. The way of producing a plan, that is, decomposing a task network is based on the unordered task decomposition [21].

Appendix B. Table of notation

See Table B.5.

References

[1] G.J. Levermore, Building Energy Management Systems: Applications to Low-energy HVAC and Natural Ventilation Control, second ed., E & FN Spon, 2000.
[2] A. De Paola, M. Ortolani, G. Lo Re, G. Anastasi, S.K. Das, Intelligent management systems for energy efficiency in buildings: A survey, ACM Comput. Surv. 47 (1) (2014) 13:1–13:38.
[3] M. Aiello, S. Dustdar, Are our homes ready for services? A domotic infrastructure based on the Web service stack, Pervasive Mobile Comput. 4 (4) (2008) 506–525.
[4] V. Degeler, L.I. Lopera Gonzalez, M. Leva, P. Shrubsole, S. Bonomi, O. Amft, A. Lazovik, Service-oriented architecture for smart environments, in: IEEE International Conference on Service Oriented Computing and Applications, 2013, pp. 99–104.
[5] R.K. Harle, A. Hopper, The potential for location-aware power management, in: International Conference on Ubiquitous Computing, ACM, 2008, pp. 302–311.
[6] European Committee for Standardization Light and lighting — lighting of work places — part 1: Indoor work places, European Standard (2011).
[7] J. Page, D. Robinson, N. Morel, J.-L. Scartezzini, A generalised stochastic model for the simulation of occupant presence, Energy Build. 40 (2) (2008) 83–98.
[8] E. Kaldeli, E.U. Warriach, A. Lazovik, M. Aiello, Coordinating the Web of services for a smart home, ACM Trans. Web 7 (2).
[9] M. Ghallab, D.S. Nau, P. Traverso, Automated Planning: Theory & Practice, Morgan Kaufmann Publishers Inc., 2004.
[10] T.A. Nguyen, A. Raspitzu, M. Aiello, Ontology-based office activity recognition with applications for energy savings, J. Ambient Intell. Humaniz. Comput. 5 (5) (2014) 667–681.
[11] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, D. Riboni, A survey of context modelling and reasoning techniques, Pervasive Mob. Comput. 6 (2) (2010) 161–180.
[12] K. Erol, J. Hendler, D.S. Nau, Htn planning: Complexity and expressivity, in: AAAI National Conference on Artificial Intelligence, 1994, pp. 1123–1128.
[13] E. Sirin, B. Parsia, D. Wu, J. Hendler, D.S. Nau, HTN planning for web service composition using SHOP2, Web Semant. 1 (2004) 377–396.
[14] F. Marquardt, C. Reisse, A. Uhrmacher, T. Kirste, A two-way approach to service composition in smart device ensembles, in: Advanced Topics in Telecommunication, 2008, pp. 49–60.
[15] I. Georgievski, Coordinating services embedded everywhere via hierarchical planning (Ph.D. thesis), Faculty of Mathematics and Natural Sciences, University of Groningen, 2015.
[16] T. Erl, SOA Principles of Service Design, Prentice Hall PTR, 2007.
[17] E. Kaldeli, A. Lazovik, M. Aiello, Domain-independent planning for services in uncertain and dynamic environments, Artificial Intelligence 236 (7) (2016) 30–64.
[18] I. Georgievski, T.A. Nguyen, M. Aiello, Combining activity recognition and AI planning for energy-saving offices, in: IEEE International Conference on Ubiquitous Intelligence and Computing, 2013, pp. 238–245.
[19] I. Georgievski, M. Aiello, Automated planning for ubiquitous computing, ACM Comput. Surv. 49 (4) (2016) 63:1–63:46.
[20] M.R. Curry, The Work in the World — Geographical Practice and the Written Word, University of Minnesota Press, 1996.
[21] I. Georgievski, M. Aiello, HTN planning: Overview, comparison, and beyond, Artificial Intelligence 222 (0) (2015) 124–156.
[22] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, D. Riboni, A survey of context modelling and reasoning techniques, Pervasive Mobile Comput. 6 (2) (2010) 161–180.
[23] E. Kaldeli, Domain-independent planning for services in uncertain and dynamic environments (Ph.D. thesis), Faculty of Mathematics and Natural Sciences, University of Groningen, 2013.
[24] B. Bonet, H. Geffner, Planning with incomplete information as heuristic search in belief space, in: International Conference on Artificial Intelligence Planning and Scheduling, 2000.
[25] A. Videla, J.J. Williams, RabbitMQ in Action, Manning, 2012.
[26] L.A. Castillo, J. Fernández-Olivares, Ó García-Pérez, F. Palao, Efficiently handling temporal knowledge in an HTN planner, in: International Conference on Automated Planning and Scheduling, 2006, pp. 63–72.
[27] I. Georgievski, HPDL: Hierarchical Planning Definition Language, JBI Preprint 2013-12-3, Uni. of Groningen (2013).
[28] M. Odersky, M. Zenger, Scalable component abstractions, SIGPLAN Not. 40 (10) (2005) 41–57.
[29] M. Odersky, L. Spoon, B. Venners, Programming in Scala: A Comprehensive Step-by-step Guide, secind ed., Artima Incorporation, 2011.
[30] L. Richardson, S. Ruby, Restful Web Services, first ed., O'Reilly, 2007.
[31] C. Hewitt, P. Bishop, R. Steiger, A universal modular ACTOR formalism for artificial intelligence, in: International Joint Conference on Artificial Intelligence, 1973, pp. 235–245.

[32] Freemeteo Weather Report Site, 2013. URL http://www.freemeteo.com.
[33] B. Setz, F. Nizamic, A. Lazovik, M. Aiello, Power management of personal computers based on user behaviour, in: International Conference on Smart Cities and Green ICT Systems, 2016, pp. 409–416.
[34] C.A. Webber, J.A. Roberson, M.C. McWhinney, R.E. Brown, M.J. Pinckard, J.F. Busch, After-hours power status of office equipment in the {USA}, Energy 31 (14) (2006) 2823–2838.
[35] Alliance to Save Energy, Energy Report 2007: United States, 2007. URL http://www.qualenergia.it/UserFiles/Files/1E%20Energy%20Report%20US.pdf.
[36] Alliance to Save Energy, PC Energy Report 2009: United States, United Kingdom, Germany, 2009. URL https://www.1e.com/EnergyCampaign/downloads/PC_EnergyReport2009-UK.pdf.
[37] L. Chiaraviglio, M. Mellia, Polisave: Efficient power management of campus pcs, in: International Conference on Software, Telecommunications and Computer Networks, 2010, pp. 82–87.
[38] T.A. Nguyen, A. Raspitzu, M. Aiello, Ontology-based office activity recognition with applications for energy savings, J. Ambient Intell. Humaniz. Comput. (2013) 1–15.
[39] A.M. Wichansky, Usability testing in 2000 and beyond, Ergonomics 43 (7) (2000) 998–1006.
[40] R. Likert, A technique for the measurement of attitudes, Arch. Psychol. 22 (140) (1932) 1–55.
[41] O. Ilghami, D.S. Nau, CaMeL: Learning method preconditions for HTN planning, in: International Conference on AI Planning and Scheduling, 2002, pp. 131–141.
[42] O. Ilghami, D.S. Nau, D.W. Aha, Learning preconditions for planning from plan traces and HTN structure, Comput. Intell. 21 (4) (2005) 388–413.
[43] N. Nejati, P. Langley, T. Konik, Learning hierarchical task networks by observation, in: International Conference on Machine Learning, ACM, 2006, pp. 665–672.
[44] N. Nejati, T. Könik, U. Kuter, A goal- and dependency-directed algorithm for learning hierarchical task networks, in: International Conference on Knowledge Capture, ACM, 2009, pp. 113–120.
[45] C. Hogg, U. Kuter, H. Muñoz Avila, Learning hierarchical task networks for non-deterministic planning domains, in: International Joint Conference on Artifical Intelligence, 2009, pp. 1708–1714.
[46] H.H. Zhuo, H. Muñoz Avila, Q. Yang, Learning hierarchical task network domains from partially observed plan traces, Artificial Intelligence 212 (0) (2014) 134–157.
[47] T. Erickson, M. Li, Y. Kim, A. Deshpande, S. Sahu, T. Chao, P. Sukaviriya, M. Naphade, The dubuque electricity portal: Evaluation of a city-scale residential electricity consumption feedback system, in: SIGCHI Conference on Human Factors in Computing Systems, ACM, 2013, pp. 1203–1212.
[48] E. von Hippel, Lead users: A source of novel product concepts, Manage. Sci. 32 (7) (1986) 791–805.
[49] T.A. Nguyen, M. Aiello, Energy intelligent buildings based on user activity: A survey, Energy Build. 56 (2013) 244–257.
[50] J. Rao, X. Su, A survey of automated web service composition methods, in: International Conference on Semantic Web Services and Web Process Composition, 2005, pp. 43–54.
[51] F. Casati, S. Ilnicki, L.-j. Jin, V. Krishnamoorthy, M.-C. Shan, Adaptive and dynamic service composition in eFlow, in: International Conference on Advanced Information Systems Engineering, 2000, pp. 13–31.
[52] Y. Feng, A. Veeramani, R. Kanagasabai, S. Rho, Automatic service composition via model checking, in: IEEE Asia-Pacific Services Computing Conference, 2011, pp. 477–482.
[53] J. Fan, S. Kambhampati, A snapshot of public web services, SIGMOD Rec. 34 (1) (2005) 24–32.
[54] S. Balzer, T. Liebig, M. Wagner, Pitfalls of owl-s: A practical semantic web use case, in: International Conference on Service Oriented Computing, 2004, pp. 289–298.
[55] K. Henricksen, J. Indulska, A. Rakotonirainy, Modeling context information in pervasive computing systems, in: Pervasive Computing, Springer, 2002, pp. 167–180.
[56] D. Nicklas, B. Mitschang, The nexus augmented world model: An extensible approach for mobile, spatially-aware applications.
[57] T. Strang, C. Linnhoff-Popien, A context modeling survey, in: Workshop Proceedings, 2004.
[58] L. Chen, I. Khalil, Activity recognition: Approaches, practices and trends, in: L. Chen, C.D. Nugent, J. Biswas, J. Hoey, I. Khalil (Eds.), Activity Recognition in Pervasive Intelligent Environments, in: Ambient and Pervasive Intelligence, vol. 4, Atlantis Press, 2011, pp. 1–31.
[59] D. Riboni, C. Bettini, COSAR: Hybrid reasoning for context-aware activity recognition, Pers. Ubiq. Comput. 15 (3) (2011) 271–289.
[60] A. Boytsov, A. Zaslavsky, K. Synnes, Extending Context Spaces Theory by Predicting Run-time Context, Springer, 2010, pp. 8–21.
[61] M. Fox, D. Long, Pddl2.1: An extension to PDDL for expressing temporal planning domains, J. Artificial Intelligence Res. 20 (1) (2003) 61–124.