Describing and Recognizing Patterns of Events in Smart Environments with Description Logics

Antonello Scalmato, Antonio Sgorbissa, and Renato Zaccaria *

June 4, 2012

Abstract

The article describes a system for context awareness in Smart Environments, which is based on an ontology expressed in Description Logics and implemented in OWL 2 EL, a subset of the Web Ontology Language which allows for reasoning in polynomial time. The approach is different from all other works in the literature since the proposed system requires only the basic reasoning mechanisms of Description Logics, i.e., subsumption and instance checking, without any additional external reasoning engine. Experiments performed with data collected in two different scenarios are described, i.e., the dataset of the CASAS Project at Washington State University, and the assisted living facility Villa Basilea in Genova, Italy.

1 INTRODUCTION

The article describes a novel approach to describe and recognize patterns of events taking place within a Smart Environment, on the basis of the information returned by a number of distributed sensors. The proposed model constitutes the basis to design a context-aware system, i.e., an intelligent system able to infer relevant information about the users and the environment, to use this information to take decisions in real-time, and to possibly adapt the system itself depending to the users’ needs and other constraints [1, 2, 3].

* A. Scalmato, A. Sgorbissa and R. Zaccaria are with the Department of Communication, Computer and System Sciences (DIST), University of Genova, Via Opera Pia 13, 16145, Genova, Italy; e-mail: {antonello.scalmato, antonio.sgorbissa, renato.zaccaria}@unige.it
The concept and the role of context awareness has become very popular in the design of intelligent systems, especially when a huge amount of data originating from different sources are available, and need to be taken into account in the decision process. Examples range from the simpler location-based services, such as proximity marketing applications which broadcast advertising content to all users within a geographic area, to Smart Home systems which pro-actively operate appliances and devices for environmental control (light, temperature, windows curtains) depending on the time of the day, the weather, energy-saving requirements, as well as on the residents location, activity, or emotions [4].

Among the different application domains, we focus on context awareness in a Smart Home. In this scenario, thanks to the miniaturization of devices and the technological advancement of wireless communication, it has become easier to retrieve information from the surrounding environment. However, in order for the available data to be of some use, there is still the open issue of how to process and merge them, to the end of providing aggregated information at a higher level of abstraction. To this purpose, many works in the literature rely on ontologies, and related languages such as OWL\(^1\), as a formal framework to represent contexts and to reason upon them. Ontologies have the property that they are easily understandable even by non-specialized user, and they provide reasoning services to infer the current context by merging primitive information. Specifically, systems relying on ontologies usually exploit both the standard inference mechanisms embedded in knowledge representation systems, e.g., class subsumption or instance checking, and user-defined rules, which must be supported by an external rule-based engines, and increase the reasoning power at the price of a lower computational efficiency.

In a similar spirit, yet with a number of differences, the approach proposed in this article relies on Description Logics [5] to build an ontology for representing and recognizing patterns of events within a Smart Environment.

The contribution of our approach is twofold. First, it allows for representing temporal relationships between events occurring in the Smart Environment, and this is achieved by relying on the syntax and the semantics of the less expressive $\mathcal{EL}^{++}$ Description Logics, a sublanguage which allows for reasoning in polynomial time [6], and can be implemented in OWL 2 EL\(^2\), a subset of the Web Ontology Language OWL 2.

Second, it recognizes patterns of events by relying exclusively on subsumption and instance checking, and it does neither require a rule-based

---

\(^1\)http://www.w3.org/TR/owl-guide/

\(^2\)http://www.w3.org/TR/owl2-overview/
system, nor any other external mechanism for reasoning. The approach is different from all the ontology-based systems for context awareness in the literature, in which context recognition (especially when dealing with temporal relationships) is left out of the main reasoning scheme of the ontology.

The two properties above are very important to allow the designer to easily describe patterns of events to be recognized. As a matter of fact, the compatibility of OWL 2 EL with the Protégé editor and related tools makes the design process easier, and accessible even to practitioners without a technological background (e.g., medical staff and caregivers within an assisted living facility).

Section 2 describes related work. Section 3 gives an overview of the system, by briefly recalling the basic principles of Description Logics. Section 4 describes the approach adopted to represent patterns of events in the ontology. Section 5 focuses on the algorithm which is used to update the ontology through sensor data. Section 6 describes experiments performed with data collected in two different scenarios, i.e., the dataset of the CASAS Project at Washington State University, and the assisted living facility Villa Basilea in Genova, Italy, where the system is currently operating 24/7. Conclusions follow.

2 Related work

Different methodological approaches to context modelling and recognition have been proposed in the literature. We start by surveying approaches which do not consider time explicitly, or consider it only minimally. Next, we discuss approaches specifically dealing with temporal issues.

2.1 Approaches which do not explicitly consider time

Systems belonging to this class are based on a number of different methodologies. The framework for ubiquitous computing Gaia proposed in [7] relies on first order logics to infer the current context starting from sensor data. In [8] a context-aware agent is presented, based on a neural network, able of learning and predicting the user’s preference on the basis of her pulse, body temperature, facial expression, location, as well as the state of the environment. Languages and systems for workflow management are proposed in [9] to describe and detect action patterns, whereas the approach described in [10] relies on Bayesian networks, and exploits causal dependencies among parameters to compensate for erroneous or missing data. In [11] a fuzzy controller is proposed to deliver context-aware services in a Smart Home, by adapting
the home behaviour to its inhabitants’ living style, and a similar solution is proposed in [12], which presents two fuzzy inference engines which have been validated through experiments in real-world scenarios. Tuples are proposed in [13] to model the user’s context, whereas context awareness in [14] is based on case-based reasoning. Other approaches exist, mostly focusing on infrastructural aspects [15, 16, 17, 18, 19]. In a slightly different spirit, instead of relying on existing methodologies and technologies, the approaches described in [20, 21, 17, 22] focus on formalizing new languages and models to describe and reason upon contexts: among them, [22] deserves a special attention in that it introduces a model which describes contexts as geometrical structures in a multidimensional space, and a context algebra which enables distributed reasoning.

More similarly to our work, some systems rely on ontologies as the central tool for context modelling and recognition. In these systems, context awareness is performed with the support of additional user-defined rules, in addition to the basic reasoning schemes (e.g., subsumption and instance checking). For example, the SOCAM architecture [23] allows for representing contexts in OWL Lite or RDF Schema, and implements a forward-chaining rule system for contexts interpretation. Similar approaches can be found in [24, 25, 26, 27]. Specifically, rules in [26] are described through the SWRL formalism, an extension to the OWL reasoning scheme, whereas CoBRA [27] uses the F-OWL rule-driven logic inference engine acting on the SOUPA [28] ontology, which has been especially designed to model relationships between places and agents. All systems relying on user-defined rules allow for a higher expressiveness but at the price of decreased system performance: specifically, procedures for rule evaluation are based on the unification of variables, which requires a higher computational cost than ontology-based reasoning.

Other approaches propose a hybrid schema integrating ontologies with different models for representation or reasoning. The authors of [29] propose MobileOntoDB, a context-aware, database-based reasoner for mobile devices: the approach uses ontologies in Description Logics, but then maps them into relational databases to be queried through SQL. Similarly, [30, 31] propose a hybrid schema involving ontologies and relational databases, and use also an external rule-based engine. Some approaches [32, 33, 34, 35] propose to integrate ontologies with Bayesian reasoning or similar probabilistic techniques, in order to deal with uncertainties. In [36] contextual information is modelled through Description Logics, by using a natural deduction style calculus [37] to reason on contexts.

Some works focus on specific aspects of a context-aware system. The work described in [38] deserves a special attention in that it focusses on learning situation models in a Smart Home; [39] deals with security and Quality
of Service; [40] focusses on failure-handling in context-aware systems; [41] envisages Shared Ontologies for Pervasive Computing (SO4PC), including modular component vocabularies to represent intelligent agents with associated beliefs, desires, and intentions; [42, 43] investigate problems related to the quality of sensor data; UCAM [44] deals with large-scale smart environments and focuses on timing performance. Application scenarios include meeting rooms [45], museums [46], ubiquitous Health Care [47, 48], airports [49], surveillance against bioterrorism [50], road traffic [51], and finally robots operating within smart spaces [52] or relying on semantic knowledge to deal with high level tasks in presence of incomplete information [53]. Differently from our work, all the approaches above are unable to represent complex temporal patterns of events.

2.2 Approaches which explicitly consider time

Similarly to the previous Section, a number of methodologies are envisaged in the literature. The authors of [54] discuss AI-techniques for dealing with temporal and spatial knowledge in Smart Homes, focusing on Allen’s algebra [55], point algebra [56], and proposing techniques to integrate temporal and spatial reasoning. In [57] an approach is proposed for activity pattern learning, recognition, and anomaly detection in the MavHome Smart Home, based on spatial reasoning and Allen Algebra. A similar approach is proposed in [58] to monitor domestic activities. Temporal logic programming is proposed in [59], and [60] proposes a model to represent and reason upon temporal contexts based on first-order logics. Finally, [61] describes the CASAS smart environment, able to find repetitive patterns in resident’s activities and to adapt to changes in those patterns. The system is composed of three main modules: a frequent and periodic activity miner (FPAM) algorithm to discover activity patterns of interest starting from raw data, a hierarchical activity model (HAM) which models these patterns depending on the underlying temporal and structural information, and finally a pattern adaptation miner (PAM) algorithm which adapts to any changes in those patterns and responds to user guidance.

Approaches to recognize temporal patterns of activities based on ontologies have been proposed as well. The TempCRM approach described in [62] relies on RDF and OWL, and uses first order logics to write user-defined rules. The system proposed in [63] defines contexts on the basis of the data returned by sensors in a given time interval: to understand the temporal evolution of the system, a rule-based system is exploited, which is external to the knowledge base itself, and therefore processes the information in a subsequent phase, after the ontology has been classified. In [64] the authors
define sequences of events, which are then classified through a Hidden Markov Model: an ontology lies on a upper layer with respect to the HMM, therefore increasing expressiveness and simplifying the definition of contexts to be monitored. The authors of [65] describe an approach for modelling complex temporal information, by presenting a methodology and a set of tools which rely on SWRL and SQWRL (the SWRL-based OWL query language). A similar approach is proposed in SS-ONT [66]: by relying on an extension of OWL, the system describes contexts within an ontology, which is then queried through the query language designed for the Semantic Web, SPARQL. The SeMaPs system described in [67] focuses on the self-management of devices, by using a number of ontologies and relying on SWRL rules to update them. Proton [68] is a Prolog reasoner which extends situation calculus [69], and is able to reason upon ontologies expressed in OWL.

Instead of simply adding temporal reasoning capability to an ontology-based system, some approaches introduce new formalisms to embed temporal constructs in the ontology. Specifically, many attempts have been done to extend Description Logics to include the temporal dimension, thus producing different Temporal Description Logics, and a wide literature in the field exist (see [70] for a survey). Works in this field focus on reasoning algorithms and related complexity results, and have been rarely applied in practical applications. In a similar spirit, [71] introduces temporal RDF, a framework to incorporate temporal reasoning into RDF which yields temporal RDF graphs, and a temporal query language for RDF. These approaches have the major drawback that they are not supported by standard tools for ontology management: since the spirit is completely different from our approach, they do not deserve a deeper discussion.

Focusing on practical applications, [72, 73] introduces TOWL, a new OWL-based temporal formalism for the representation of time, change, and state transitions: however, in spite of the novelty of the formalism, the approach still uses Jena rule to check relationships between temporal intervals.

All the approaches above are different from the work described in this article, since our approach allows for modelling and recognizing complex temporal patterns of events by relying exclusively on subsumption and instance checking, i.e., the basic mechanisms for reasoning on the ontology, and does not require any additional inferential mechanisms.

3 System overview

In this Section, the basic principles of Description Logics are briefly recalled, by limiting the analysis to those aspects which are more relevant for the
following discussion (for a deeper analysis refer to [3]). Then, the architecture of the proposed system is introduced, by outlining information flows to and from the ontology.

3.1 Description Logics

Description Logics is a formalism for knowledge representation that describes a given domain by defining relevant concepts (the Terminological Box, also referred to as TBox), and then asserting properties of individuals which are instances of those concepts (the Assertional Box, also referred to as ABox). Description Logics owes its success to the fact that it provides services for reasoning about represented concepts and individuals to infer hidden knowledge, and both the representation and the reasoning services are based on a well-defined formal semantics.

The TBox consists of concept descriptions, which allow the user to iteratively define concepts on the basis of other (possibly atomic) concepts. For instance, given that Person and Address are atomic concepts, it is then possible to define

\[
\text{Resident} \equiv \text{Person} \sqcap \exists \text{hasDomicile.Address.} \tag{1}
\]

In the previous expression \text{hasDomicile} is a role, i.e., a binary relation between concepts. Intuitively, the concept \text{Resident} represents a person who is characterized by having a domicile, i.e., (1) associates a symbolic name to a complex description. The TBox is referred to as being definitorial if it does not contain cycles, i.e., if all defined concepts can be ultimately described in terms of primitive (not defined) concepts.

The ABox consists of individuals, which are given a name and whose properties can be asserted. For instance, by referring to the example above,

\[
\begin{align*}
\text{Person(DUDLEY),} \\
\text{Address(PRIVETDRIVE4),} \\
\text{hasDomicile(DUDLEY, PRIVETDRIVE4),}
\end{align*}
\tag{2}
\]

assert properties about the named individuals DUDLEY and PRIVETDRIVE4. Intuitively, the former two assertions (also referred to as concept assertions) mean that “Dudley” is a person, and “Privet Drive number 4” is an address. The latter assertion (also referred to as a role assertion) means that “Dudley” lives at “Privet Drive number 4”.

Reasoning in the TBox is performed by checking relationships involving concepts and roles, i.e., satisfiability, equivalence, disjointness, all of which can be ultimately reduced to checking the subsumption relationship between
concepts. A formal explanation of subsumption would require to introduce
the semantics of Description Logics: informally, a concept $C$ subsumes an-
other concept $D$ when the description of the former is more general than or
equal to the latter. For instance, given that the following inclusion is asserted
in the TBox

$$\text{Wizard} \sqsubseteq \text{Person},$$

a non-atomic concept corresponding to the description

$$\text{Wizard} \sqcap \exists \text{hasDomicile.Address}$$

is subsumed by the concept Resident. The process of finding, for all concepts
in the TBox, the most specific subsuming concept in the taxonomy, is also
referred to as classification.

Reasoning in the ABox includes consistency checking, instance checking
(to check if an individual is an instance of a given concept), retrieval (to find
all individuals which are instances of a given concept), realization (to find
the most specific concepts, with respect to the subsumption relationship, of
which an individual is an instance). Among the others, instance checking
constitutes the main reasoning scheme for context inference in our model. In
the example above, given that the ABox includes the additional assertions

$$\text{Wizard(HARRY)},$$

$$\text{hasDomicile(HARRY, PRIVETDRIVE4)},$$

it can be inferred that HARRY is an instance of Resident.

A number of constructors are available to build complex descriptions in
the TBox starting from atomic concepts: the subset of constructors available
can vary from case to case, and determines the expressiveness of a specific
sublanguage of Description Logics. Examples range from the attributive
language $\mathcal{ALC}$, which allows for the negation of atomic concepts, concept
intersection, universal value restrictions, and limited existential quantification,
to the more complex $\mathcal{SHOIN}(D)$ Description Logics$^3$, which allows for the
negation of non atomic concepts ($\mathcal{C}$), union of concepts ($\mathcal{U}$), full existential
quantification ($\mathcal{E}$), role hierarchies ($\mathcal{H}$), usage of nominals in concept defini-
tions ($\mathcal{O}$), inverse on roles ($\mathcal{I}$), role cardinality restrictions ($\mathcal{N}$), and usage of
basic data types ($\mathcal{D}$). The more expressive is the language, the higher is the
computational cost of reasoning procedures. Specifically, it has been shown
that, in presence of universal value restrictions, subsumption is NP-hard in
the worst case even with the simplest form of acyclic TBoxes [74].

$^3$ $\mathcal{S}$ is an abbreviation for $\mathcal{ALC}$, where $\mathcal{C}$ stands for non-atomic concept negation, a
description logics which can express also union $\mathcal{U}$ and full existential quantification $\mathcal{E}$.
3.2 System architecture

In the following, the term context refers to spatially distributed events taking place in the same time interval, whereas situation refers to a temporally related sequence of contexts.

In order to define an ontology of contexts and situations to be monitored in the environment, the system described here is based on $\mathcal{EL}^{++}$ [6], a lightweight Description Logics which supports full existential quantification and datatypes, but not universal value restriction: as a consequence of the limitations on its expressiveness, subsumption in $\mathcal{EL}^{++}$ can be checked in polynomial time with respect to the size of the ontology. The language is supported by OWL 2 under the profile OWL 2 EL, thus making possible to use Protégé for ontology editing.

Table 1: Concepts and roles provided by the system

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>hasFluent</td>
</tr>
<tr>
<td>Interval</td>
<td>hasBeginsAt</td>
</tr>
<tr>
<td>LongerThan</td>
<td>hasPredecessor</td>
</tr>
<tr>
<td>ShorterThan</td>
<td></td>
</tr>
<tr>
<td>FartherThan</td>
<td></td>
</tr>
<tr>
<td>CloserThan</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1 shows an overview of the system. Two main blocks are visible in the figure: an ontology, which is composed of a TBox and an ABox, and a reasoner, which operates on the ontology in order to infer the current contexts and situations. Inputs from the user and from sensors are used to update the ontology when required.

Context and situation recognition works in two phases, one which is performed off-line by the user, and the other performed on-line by the system.
itself whenever new sensor data are available.

The off-line phase basically requires the user to populate the TBox with proper concept definitions which depend on the specific application domain: sensors, contexts and situations to be monitored. Specifically, the system provides the user with a small set of concepts and roles (Table 1) which must be used as a basis to define application-specific concepts. In order to allow for context and situation recognition through ontology-based reasoning, the system requires a definitional TBox where all user-defined concepts can be ultimately described in terms of the concepts and roles made available by the system.

The on-line phase consists of two sub-phases:

- a knowledge acquisition phase, which populates the ABox with a set of new individuals which reflect the current state of the system depending on sensors;

- a reasoning phase, which infers if a given individual is an instance of any concept which corresponds to a context or a situation deemed relevant in the application domain.

In the next Sections these two phases are described in details.

4 Off-line phase: populating the TBox

The user must describe the application domain in the TBox according to a three-layered organization: the lower layer describes sensor data, the middle layer aggregates sensor data which are concurrently available to infer the current contexts, the upper layer expresses mutual temporal relationships between contexts to describe situations.

4.1 Lower layer: sensors

In the following, a sensor do not necessarily corresponds to the usual definition of “a device that measures a physical quantity and converts it into a signal”. As a matter of fact, the definition is extended to consider any actual or virtual source of information at different levels of abstractions, ranging from clocks returning the time of the day, to “elevator sensors” returning the current state of an elevator.

Sensors can produce different types of outputs, e.g., boolean, integer, or discrete sets of values. In order to easily integrate all these data in the ontology, we introduce the concept of “state of a sensor” as the most recent
output returned by that sensor. Then, each state of a sensor which is relevant in the system corresponds to a different concept in the ontology. For instance, when using a Passive Infrared Sensor (PIR) to detect the presence of people, the state is a binary value \{On, Off\}. When using a temperature sensor, a quantization is performed on the output temperature levels, and the state is defined as a value in the discrete set \{Cold, Warm, Hot\}. Then, to model the possible states of a PIR sensor located, say, in the kitchen, the user must refer to the atomic concept Sensor (Table 1), and define

\[
\begin{align*}
\text{Pir} & \sqsubseteq \text{Sensor}, \\
\text{Pir}_\text{Kitchen}_\text{On} & \sqsubseteq \text{Pir}, \\
\text{Pir}_\text{Kitchen}_\text{Off} & \sqsubseteq \text{Pir}.
\end{align*}
\]

This makes possible to instantiate individuals on the basis of the returned sensor data, therefore anchoring representations in the ontology to perceptions in the real world. Suppose that the PIR located in the kitchen is detecting a person: this information can be added to the ontology by asserting a proper individual the ABox, i.e.,

\[
\text{Pir}_\text{Kitchen}_\text{On}(\text{PIR1}).
\]

**Remark 1** The reader may notice that \(\mathcal{EL}^{++}\) allows for a more expressive solution to describe sensor states, starting from measurements expressed through basic datatypes (e.g., integer or string):

\[
\begin{align*}
\text{Pir}_\text{Kitchen} & \equiv \text{Pir} \sqcap \exists \text{hasId}(=, \text{"Kitchen"}), \\
\text{Pir}_\text{Kitchen}\_\text{On} & \equiv \text{Pir}_\text{Kitchen} \sqcap \exists \text{hasState}(=, 1),
\end{align*}
\]

which then allows one to assert in the ABox

\[
\begin{align*}
\text{Pir}(\text{PIR1}), \\
\text{hasId}(\text{PIR1}, \text{"Kitchen"}), \\
\text{hasState}(\text{PIR1}, 1),
\end{align*}
\]

which is automatically recognized as an instance of \text{Pir}_\text{Kitchen}_\text{On}. However, this latter solution is more computationally expensive during reasoning.

### 4.2 Middle layer: contexts

The middle layer deals with contexts, i.e., configurations of sensors states which deserve to be monitored and recognized in the environment. To deal with contexts (and then, by adding the temporal dimension, with situations), the main concept in the ontology (Table 1) is
\[
\text{Interval} \equiv \exists \text{hasFluent.Sensor} \sqcap \\
\exists \text{hasBeginsAt.Integer} \sqcap \\
(\text{hasPredecessor} \circ \text{hasPredecessor} = \text{hasPredecessor}).
\]

The \text{hasFluent} role (Table 1) establishes a relationship between individuals of type \text{Interval} and the corresponding individuals which denote sensor states holding in that interval (sensor states are referred to as "fluents" since they correspond to conditions that can change over time). The \text{hasBeginsAt} role (Table 1) is used to store the first time instant when that configuration of sensor states became true.

The role-value map \((\text{hasPredecessor} \circ \text{hasPredecessor} = \text{hasPredecessor})\) is a concept constructor which is allowed by \(\mathcal{E}L^{++}\): it is required to introduce transitivity when defining situations as sequences of intervals (see Section 4.3), and it can be ignored for the moment.

A context can now be described through the \text{Interval} concept, with additional constraints on its roles to specify that a subset of sensors must be in a given state for that context to be inferred. This can be better explained through an example: consider the context "the user has taken the medicine dispenser",

\[
\text{Dispenser\_Taken} \equiv \text{Interval} \sqcap \\
\exists \text{hasFluent.Item\_Dispenser\_On} \sqcap \\
\exists \text{hasFluent.Pir\_Kitchen\_On}.
\]

The context \text{Dispenser\_Taken} requires two sensors to be in a prescribed state. Specifically, the role restrictions on \text{hasFluent} specifies that the "item sensor" which monitors whether the dispenser is in its docking position or not must be \text{On} (not in docking position), and that the PIR sensor which detects people in proximity of the dispenser must be \text{On} as well (at least one person detected).

Suppose now that the following individuals are asserted at time \(t_1\) in the ABox which already contains the definition in (7),

\[
\begin{align*}
\text{Item\_Dispenser\_On}(\text{ITEM1}), \\
\text{Interval}(E1), \\
\text{hasBeginsAt}(E1, T1), \\
\text{hasFluent}(E1, \ldots) & \quad \text{other sensor states,} \\
\text{hasFluent}(E1, \text{PIR1}), \\
\text{hasFluent}(E1, \text{ITEM1}), \\
\text{hasFluent}(E1, \ldots) & \quad \text{other sensor states.}
\end{align*}
\]

Then, when asking for the most specific concept of which \(E1\) is an instance, the inferential mechanism necessarily returns \text{Dispenser\_Taken}.
Remark 2  Context recognition through instance checking has the effect of filtering out information which is not relevant: among the others, the state of all sensors which do not appear explicitly in the context definition (11), but fill the hasFluent role of the individual E1 (12).

To describe a context as a configuration of sensor states whose duration is longer or shorter than a given time interval, concepts modelling intervals with different durations are derived from the concepts LongerThan and ShorterThan, which – on their turn – are derived from Interval. Specifically, the system makes available (Table 1) a set of concepts recursively derived from LongerThan, e.g.,

\[
\begin{align*}
\text{LongerThan} & \sqsubseteq \text{Interval}, \\
\text{LongerThan1minute} & \sqsubseteq \text{LongerThan}, \\
\text{LongerThan5minutes} & \sqsubseteq \text{LongerThan1minute}, \\
\text{LongerThan10minutes} & \sqsubseteq \text{LongerThan5minutes} \\
\cdots 
\end{align*}
\]

(13)

The hierarchy above allows the user to express the intuitive notion that an interval longer than 10 minutes is necessarily an interval longer than 5 minute, and so on. Symmetrically, the system makes available (Table 1) concepts derived from ShorterThan as

\[
\begin{align*}
\text{ShorterThan} & \sqsubseteq \text{Interval}, \\
\cdots \\
\text{ShorterThan10minutes} & \sqsubseteq \text{ShorterThan30minutes}, \\
\text{ShorterThan5minutes} & \sqsubseteq \text{ShorterThan10minutes}, \\
\text{ShorterThan1minute} & \sqsubseteq \text{ShorterThan5minutes}, \\
\end{align*}
\]

(14)

which allows the user to express the intuitive notion that an interval shorter than 10 minutes is necessarily an interval shorter than 30 minute, and so on.

Starting from the concepts above, the user is allowed to define, for example, the following context corresponding to a “user performing a long phone call”,

\[
\begin{align*}
\text{MakingLongPhonecall} & \equiv \text{Interval} \sqcap \text{LongerThan10minutes} \\
& \exists \text{hasFluent.Phone_Active},
\end{align*}
\]

(15)

which describes a sensor configuration in which the phone receiver is picked up for longer than 10 minutes. To verify how the context in (15) can be recognized through instance checking, suppose that the following individuals
are inserted at time $t_2$ in the ABox which already contains the individual in (7), (12)

\[
\begin{align*}
\text{Phone\_Active(}\text{PHONE1}), \\
\text{Interval(}\text{E2}), \\
\text{hasBeginsAt(}\text{E2}, \text{T2}), \\
\text{hasFluent(}\text{E2, \ldots}) & - \text{other sensor states,} \\
\text{hasFluent(}\text{E2, PHONE1}), \\
\text{hasFluent(}\text{E2, \ldots}) & - \text{other sensor states.}
\end{align*}
\]

(16)

At any time, additional properties can be asserted concerning $E2$ to express its duration: specifically, if at time $t_3$ the difference $t_3 - t_2$ is bigger then, say, 20 minutes, the ontology can be enriched with the assertion

\[
\text{LongerThan20minutes(}\text{E2}),
\]

(17)

which allows $E2$ to be correctly classified as an instance of MakingLongPhonecall.

**Remark 3** As a consequence of how concepts are defined in (13), one is always allowed to add new assertions concerning an individual which is instance of LongerThan as time passes. For example, if at time $t_4$ the difference $t_4 - t_2$ is bigger then 30 minutes, it is possible to enrich the ABox with the assertion LongerThan30minutes(E2) without violating monotonicity. The same does not hold with instances of ShorterThan (14).

Finally contexts can be iteratively defined starting from other contexts, e.g.,

\[
\begin{align*}
\text{Water\_Taken} & \equiv \text{Interval} \sqcap \exists \text{hasFluent.Tap\_Open}, \\
\text{Medicine\_Taken} & \equiv \text{Dispenser\_Taken} \sqcap \text{Water\_Taken},
\end{align*}
\]

(18)

thus simplifying descriptions and improving readability.

### 4.3 Upper layer: situations

By expressing temporal relationships between contexts, it is possible to define situations. Specifically, situations are represented through the Interval concept introduced in (10), by specifying a temporal precedence relationship through the hasPredecessor role (Table 1), and by possibly putting additional quantitative constraints. Notice that, intuitively, one would expect the hasPredecessor role to be transitive, meaning that, if interval $e_1$ is a predecessor of $e_2$ and $e_2$ is a predecessor of $e_3$, then $e_1$ is a predecessor of $e_3$. In [75] it is shown that restricted role-value maps in the form $r_1 \circ r_2 \sqsubseteq r_3$ can be used in $\mathcal{EL}^{++}$ to express transitivity while leaving the reasoning problem polynomial.
Practically, the role-value map \((\text{hasPredecessor} \circ \text{hasPredecessor} = \text{hasPredecessor})\) is a concept constructor, which determines the following property to recursively hold for all individuals \(e_i\) which are instances of \(\text{Interval}\): if the role \(\text{hasPredecessor}\) of an individual \(e_1\) is filled by \(e_2\), and the role \(\text{hasPredecessor}\) of \(e_2\) is filled by \(e_3\), then the role \(\text{hasPredecessor}\) of \(e_1\) is filled by \(e_3\).

Then, a situation is basically a list of contexts, temporally ordered from the more recent to the older one. For instance,

\[
\text{PreparingMeal} \equiv \text{Microwave\_Closed} \sqcap \\
\exists \text{hasPredecessor}. (\text{Microwave\_Open} \sqcap \\
\exists \text{hasPredecessor}. \text{Pot\_Taken}),
\]

where \(\text{Microwave\_Closed}\), \(\text{Microwave\_Open}\), \(\text{Pot\_Taken}\) have been properly defined as contexts by the user.

The reader may notice that, in addition to temporal precedence, it would be important to represent situations which correspond to the temporal overlapping of contexts, yet Table 1 does not contain any role to express such relationship. The reason is straightforward: since a context is represented as a time interval where a given configuration of sensors holds, the overlapping relationship can be simply expressed by defining a situation as the conjunction of two contexts (i.e., a time interval where the sensor configurations corresponding to both contexts hold). As a matter of fact, the definition of \(\text{Medicine\_Taken}\) (18) can be interpreted as a situation of this type.

As an example of situation recognition, suppose now that, at time \(t_6\), the following individuals have been added in the ABox which already contains (7), (12), (16), (17):

\[
\text{Interval}(E3), \text{hasPredecessor}(E3, E2), \\
\text{Interval}(E4), \text{hasPredecessor}(E4, E3), \\
\text{Interval}(E5), \text{hasPredecessor}(E5, E4), \\
\text{Interval}(E6), \text{hasPredecessor}(E6, E5), \\
\text{hasBeginsAt}(E6, T6),
\]

and that the individuals above have the following properties

\[
\text{Pot\_Taken}(E3), \\
\text{Medicine\_Taken}(E4), \\
\text{Microwave\_open}(E5), \\
\text{Microwave\_closed}(E6),
\]

then, thanks to the role-value map (10) expressing transitive properties, \(E6\) would also be classified as

\[
\text{PreparingMeal}(E6),
\]
by automatically filtering out E4 and all other individuals which do not appear explicitly in the definition of the situation (19) to be recognized.

The user is also allowed to put quantitative constraints on the maximum or the minimum time which must elapse between two contexts for inferring a given situation. To this end, the system exploits the possibility offered by $\mathcal{EL}^{++}$ to define concrete domains, and introduces a set of concepts derived from FartherThan and CloserThan (Table 1):

\[
\begin{align*}
\text{FartherThan} & \sqsubseteq \text{Interval}, \\
\text{FartherThan}1\text{minute} & \equiv \text{FartherThan} \cap \\
& \exists \text{hasBeginsAt}. (<, t - 1\text{min}), \\
\text{FartherThan}5\text{minute} & \equiv \text{FartherThan} \cap \\
& \exists \text{hasBeginsAt}. (<, t - 5\text{min}), \\
\ldots \\
\text{CloserThan} & \sqsubseteq \text{Interval}, \\
\text{CloserThan}1\text{minute} & \equiv \text{CloserThan} \cap \\
& \exists \text{hasBeginsAt}. (>, t - 1\text{min}), \\
\text{CloserThan}5\text{minute} & \equiv \text{CloserThan} \cap \\
& \exists \text{hasBeginsAt}. (>, t - 5\text{min}), \\
\ldots
\end{align*}
\]

(23)

The concepts above allow the user to describe, for instance, the situation

\[
\begin{align*}
\text{HavingMeal} & \equiv \text{AtTable} \cap \exists \text{hasPredecessor}. \\
(\text{PreparingMeal} \cap \text{CloserThan20minutes})
\end{align*}
\]

(24)

which filters out the case of a user who sits down at the table a long time after having prepared the meal, e.g., for having her afternoon tea.

The usage of concrete domains in (23) deserves a deeper discussion. As a matter of fact, the value restriction on the hasBeginsAt role is written in italics since it must be re-computed at every time step $t$, therefore requiring to update in the TBox all definitions of concepts derived from FartherThan and CloserThan (this is done by the system itself in the online phase).

For instance suppose that, at time $t_7$, the individual \text{Interval}(E7) can be inferred as being an instance \text{AtTable}(E7). Suppose also that time is expressed in minutes with respect to an absolute time frame, and $t_7 = 127\text{min}$: the concept definitions in (23) is automatically updated by setting

\[
\begin{align*}
\text{CloserThan20minute} & \equiv \text{CloserThan} \cap \\
& \exists \text{hasBeginsAt}. (>, 107).
\end{align*}
\]

(25)

Then the individual E6, which is an instance of \text{PreparingMeal} (22), is also an instance of \text{CloserThan20minute} if and only if it has started at time $t_6 >$
107min. If this condition holds, E7 is inferred as being an instance of HavingMeal, otherwise it is not.

**Remark 4** Updating definition (23) at every iteration t necessarily produces an ontology which does not increase monotonically in time, thus requiring to re-consider the results of inferences produced in previous iterations. This appears to be an acceptable compromise to support quantitative reasoning about time in absence of a rule-based inference engine. Moreover, for every snapshot of the ontology at a given time t, monotonicity is an intrinsic characteristic of \( \mathcal{EL}^{++} \), thus guaranteeing decidability as well as computational efficiency.

Finally, situations can be iteratively defined starting from other situations and contexts, e.g.,

\[
\text{FollowingPrescriptions} \equiv \text{Medicine}_\text{Taken} \cap \exists \text{hasPredecessor.} \\
(Having\text{Meal} \cap \text{CloserThan30minutes}) \tag{26}
\]

to improve readability.

### 4.4 Comparison with Allen's Algebra

It is worth spending a word on the relationships between the approach proposed in this work to describe temporal patterns of events, and the Allen’s interval algebra (which is adopted, among the others, by the approach proposed in [58]).

Allen’s algebra introduces a set of 13 basic relations which can involve couples of time intervals, and a set of composition rules which allows inferring the temporal relations between any two intervals starting from basic relations [55].

As a matter of fact, the Interval concept in (10) allows the user to encode only a subset of Allen’s relations when defining contexts and situations: however, this is a choice, and not an intrinsic limitation of the approach. If required, one is allowed to redefine the Interval concept to potentially express all the composition rules required in Allen’s algebra by using role-value maps. For instance, according to Allen’s algebra, if interval E1 meets E2 (e.g., E2 starts exactly when E1 finishes) and E2 is during E3, it should be possible to infer, through composition rules, that one of the three relations must hold: E1 overlaps E3, E1 is during E3, or E1 starts E3. The composition rule above can be expressed by updating (10) as follows:
Interval ≡ ∃hasFluent.Sensor\(\sqcap\)
\[
\ldots
\]
(\text{meets} \circ \text{during} = \text{overlaps})\(\sqcap\) \tag{27}
(\text{meets} \circ \text{during} = \text{during})\(\sqcap\)
(\text{meets} \circ \text{during} = \text{starts})

Then, given an ABox with the following assertions

\[
\text{Interval}(E_1), \text{Interval}(E_2), \text{Interval}(E_3), \\
\text{meets}(E_1, E_2), \text{during}(E_2, E_3),
\]

the following role assertions are implied:

\[
\text{overlaps}(E_1, E_3), \text{during}(E_1, E_3), \text{starts}(E_1, E_3),
\]

which are taken into account by the reasoner while performing instance checking.

Of course, representing all composition rules in Allen’s algebra necessarily reduces computational efficiency and – in our view – is not necessarily required. Then, in the following, we refer to the simpler definition in (10): through experiments we will demonstrate that, by considering also quantitative constraints in (13), (14), (23), the system exhibits the required expressiveness to represent a huge set of real-world scenarios.

5 On-line phase: making assertions in the ABox

5.1 Algorithm description

The on-line phase deserves a specific attention. The rationale is that of representing, as time passes, individual time intervals \(e\) in the ABox, by explicitly stating the starting time \(t\) of each interval \(e\) through role assertions of type \text{hasBeginsAt}(e, t), the sensor states \(s\) which hold in each interval \(e\) through role assertions of type \text{hasFluent}(e, s), precedence relationships between pairs of individuals \(e_c, e_{c-1}\) through role assertions of type \text{hasPredecessor}(e_{c-1}, e_c), as well as other properties.

As the reader may suspect, since the system does not rely on external rule-based mechanisms for context inference, the complexity of reasoning is partially moved to Algorithm 1 which updates the ontology. However, this algorithm does not perform any reasoning: it simply implements a policy for making assertions in the ABox at every time step \(t\) (as well as minimal updates in the TBox).
The algorithm is described in details, which are important to understand internal mechanisms, but should not be essential to read the rest of the paper. The following definitions are in order.

- \( \{S_i\}, \ i = 1 \ldots N \), is a set of concepts, with \( S_i \) representing the \( i^{th} \) sensor;
- \( \{S_{ij}\}, \ i = 1 \ldots N, j = 1 \ldots M_i \), is a set of concepts, with \( S_{ij} \) representing the \( j^{th} \) state of the \( i^{th} \) sensor;
- \( \mathbb{L} \) is a list of individuals, all of them being instances of a concept in the set \( \{S_{ij}\} \);
- \( \mathbb{E} \) is a list of individual instances of \texttt{Interval}, ordered with the most recently asserted individual at the head;
- \( \mathcal{T} \) is the TBox, \( \mathcal{A} \) is the ABox.

Algorithm 1 works as follows. Lines 1 to 6 initialize the system: specifically, Line 1 initializes \( \mathcal{A} \) as well as the two lists \( \mathbb{L}, \mathbb{E} \); Lines 2 to 6 add a concept assertion for each sensor state which the user has deemed relevant to define a context or situation in \( \mathcal{T} \).

Lines 7 to 49 define the main cycle, and are repeatedly executed while the system is working. Line 8 reads the current time at every iteration. Lines 9 to 41 are executed whenever a sensor \( i \) changes its state, say from the \( j^{th} \) to the \( l^{th} \); they basically assert new individual intervals \( e \) in \( \mathcal{A} \) and properly fill their role as required to reflect the new configuration of sensor states. Lines 42 to 48 are executed periodically: they update the temporal duration of every individual interval \( e \), check if \( e \) is an instance of some concept \( D \) corresponding to a situation which the users wants to monitor, and update the definition of concepts in \( \mathcal{T} \) derived from \texttt{CloserThan} or \texttt{FartherThan}, which are used to express quantitative temporal constraints (23). Notice that, for each \( D \) of which \( e \) is an instance, Line 45 adds a concept assertion \( D(e) \) in \( \mathcal{A} \).

More in details, Lines 30 to 40 are executed only when the new state \( l \) of the \( i^{th} \) sensor is among the states which have a corresponding individual \( S_{il}(s) \) in \( \mathcal{A} \) (remember that, according to Lines 2 to 6, there are states with no correspondence in \( \mathcal{A} \)). Line 31 inserts the individual sensor state \( s \) into the list \( \mathbb{L} \) of states which currently hold. Lines 32 to 39 assert a new individual interval \( e_c \) as well as its properties: among the others, a precedence relationship between \( e_{c-1} \) and \( e_c \) expressed through the \texttt{hasPredecessor} role.
Algorithm 1 Update ABox

Require: $\mathcal{T}$, $\mathcal{A}$
Ensure: $\mathcal{T}$, $\mathcal{A}$

1: initialize $c = 0$, $\mathcal{A} = \text{Interval}(e_c)$, $\mathcal{L} = \emptyset$, $\mathcal{E} = e_c$
2: for all concept $S_{ij}$ modelling a sensor state do
3: if $S_{ij}$ is used in $\mathcal{T}$ to define a context or situation then
4: assert $S_{ij}(s)$ in $\mathcal{A}$
5: end if
6: end for
7: while true do
8: get the current time $t$
9: if sensor $i$ changes from state $j$ to state $l$ then
10: if $S_{ij}(s) \in \mathcal{A}$ then
11: remove $s$ from $\mathcal{L}$
12: get $e$ at the head of $\mathcal{E}$
13: while $e \neq \emptyset$ and hasFluent($e$, $s$) $\in \mathcal{A}$ do
14: get the individual $e'$ following $e$ in $\mathcal{E}$
15: if $e' \neq \emptyset$ and hasFluent($e'$, $s$) $\in \mathcal{A}$ then
16: increment the counter $c$
17: assert Interval($e_c$) in $\mathcal{A}$
18: for all $R(e, x) \in \mathcal{A} \neq \text{hasFluent}(e, s)$ do
19: assert $R(e_c, x)$ in $\mathcal{A}$
20: end for
21: substitute $e$ with $e_c$ in $\mathcal{E}$
22: else
23: remove $e$ from $\mathcal{E}$
24: end if
25: assert $C(e)$ in $\mathcal{A}$, with $C \sqsubseteq \text{ShorterThan}$
26: for all situations $D$ of which $e$ is an instance, assert $D(e)$
27: $e = e'$
28: end while
29: end if
30: if $S_{il}(s) \in \mathcal{A}$ then
31: insert $s$ in $\mathcal{L}$
32: increment the counter $c$
33: assert Interval($e_c$) in $\mathcal{A}$
34: assert hasBeginsAt($e_c$, $t$) in $\mathcal{A}$
35: for all $s$ in $\mathcal{L}$ do
36: assert hasFluent($e_c$, $s$) in $\mathcal{A}$
37: end for
38: assert hasPredecessor($e_c, e_{c-1}$) in $\mathcal{A}$
39: put $e_c$ at the head of $\mathcal{E}$
40: end if
41: end if
42: if enough time has passed then
43: for all $e$ in $\mathcal{E}$ do
44: etc.
45: end for
46: end if
Figure 2: The graph describes how individuals are inserted into (rising edge) and removed from (falling edge) \( \mathbb{L} \) and \( \mathbb{E} \) as time passes.

Line 39 adds \( e_c \) at the head of \( \mathbb{E} \), the list of individual intervals which correspond to sensor configurations holding at time \( t \) (intuitively, intervals which have not ended yet).

Symmetrically, lines 10 to 29 are executed only when the past state \( j \) of the \( i^{th} \) sensor has a corresponding individual \( S_{ij}(s) \) in \( \mathcal{A} \). Line 11 removes \( s \) from the list \( \mathbb{L} \) of current sensor states. Lines 13 to 28 browse \( \mathbb{E} \), from the most recently added interval to the oldest one, in order to find all individuals \( e \) such that \texttt{hasFluent}(\( e, s \)) is asserted in \( \mathcal{A} \). Then, for every \( e \), if \( e \) is the individual which was asserted at the very time instant \( t' < t \) when the \( i^{th} \) sensor had switched to the \( j^{th} \) state (i.e., \( e \) is the least recently added individual in \( \mathbb{E} \) which has \( s \) as a filler), Line 23 simply removes \( e \) from \( \mathbb{E} \). If \( e \) was asserted at a later time \( t'' \), with \( t' < t'' < t \), Line 21 substitutes \( e \) with a new interval \( e_c \), which has the same properties as \( e \) (including the starting time expressed through the \texttt{hasBeginAt} role) with the only exception that \texttt{hasFluent}(\( e_c, s \)) is not asserted in \( \mathcal{A} \). In both cases, when \( e \) is removed from \( \mathbb{E} \), Line 25 updates the temporal duration of \( e \), and Line 26 checks if \( e \) is an instance of some concept \( D \) corresponding to a situation which the users wants to monitor, and possibly adds a concept assertion \( D(e) \) in \( \mathcal{A} \).

To better illustrate the behaviour of the algorithm, consider the example in Figure 2, which describes how individuals are inserted (rising edge) and removed (falling edge) from \( \mathbb{L} \) and \( \mathbb{E} \) as time passes. Sensors 1, 2, and 3 are initially in states which are not modelled in \( \mathcal{A} \). At time \( t_1 \), sensor 1 enters a new state which has been asserted in \( \mathcal{A} \) as \( \texttt{S}11(s11) \), and
lines 30 to 40 are executed: $s_{11}$ is inserted in $\mathbb{L}$, and a new individual $\text{Interval}(E1)$ is asserted in $\mathcal{A}$ and added at the head of $E$, by properly asserting $\text{hasBeginsAt}(E1, t1)$ to express its starting time and $\text{hasFluent}(E1, s_{11})$ to express the corresponding configuration of sensor states. Something similar happens in $t_2$ and $t_3$, therefore producing $\text{Interval}(E2)$ and $\text{Interval}(E3)$, which are asserted in $\mathcal{A}$ together with their properties $\text{hasBeginsAt}(E2, t2)$, $\text{hasFluent}(E2, s_{11})$, $\text{hasFluent}(E2, s_{21})$, $\text{hasBeginsAt}(E3, t3)$, $\text{hasFluent}(E3, s_{11})$, $\text{hasFluent}(E3, s_{21})$, $\text{hasFluent}(E3, s_{31})$, and added at the head of $E$. In order to express the temporal relationships between intervals, the system asserts $\text{hasPredecessor}(E2, E1)$, $\text{hasPredecessor}(E3, E2)$.

Notice that, when a new individual $\text{Interval}(e)$ is asserted, its $\text{hasFluent}$ role is filled with all individuals $s_{ij}$ in $\mathbb{L}$, corresponding to the current sensor states. Periodically (Lines 42 to 48), the just added individual will be considered (Line 45) to check if it is an instance of a context or a situation defined as relevant by the user. For instance, in $t_3$, $E3$ would be classified as an instance of a hypothetical context $C_1$ which requires sensors 1, 2, and 3 to be in state 1. Additionally, according to the subsumption relationship, $E3$ would also be classified as an instance of a more general context $C_2 \supseteq C_1$ which requires only, e.g., sensor 1 and 3 to be in state 1 without any constraint on the state of sensor 2.

At time $t_4$, sensor 1 switches from state 1 to another state (which, in this example, has not a corresponding individual in $\mathcal{A}$). Then, Lines 10 to 29 have the effect of removing from $E$ all individuals whose $\text{hasFluent}$ role is filled with $s_{11}$, i.e., individuals $E1$, $E2$, $E3$ which have been asserted after $t_1$. However, this is not sufficient: as a matter of fact, after removing $E1$, $E2$, $E3$ the list $E$ is empty, but sensor 2 is still in state 1 (starting from $t_2$) and sensor 3 is in state 1 (starting from $t_3$). To model these configurations of sensor states, Line 21 substitutes $E2$ with a new individual $E4$ which has the same properties of $E2$ (including the starting time) with the exception of $\text{hasFluent}(E4, s_{11})$. A similar substitution is operated on $E3$ by inserting a new individual $E5$.

**Remark 5** The way in which individual intervals are added and removed from $E$, which may appear complicated, allows the system to explicitly represent and reason on the temporal duration of a given configuration of sensor states, by making concept assertions $C(e)$, with $C \sqsubseteq \text{LongerThan} (13)$ or $C \sqsubseteq \text{ShorterThan} (14)$. Notice also that concept assertions of the latter type can only be made when $e$ is removed from $E$ (Line 25), i.e., when the configuration of sensor states which corresponds to that interval is no more valid.

**Remark 6** Explicitly adding concept assertions corresponding to recognized contexts and situations (Lines 26 and 46) has multiple purposes, among which
that of allowing the user to express situations as sequences of intervals with quantitative temporal constraints. For instance, recognizing an interval \( e \), at time \( t \), as an instance of FollowingPrescription requires that an interval \( e' \) has been recognized as an instance of HavingMeal at a time \( t' \) not sooner than 30 minutes before \( t \) (26), and an interval \( e'' \) has been recognized as an instance of PreparingMeal at a time \( t'' \) not sooner than 20 minutes before \( t' \) (24). However, the definition of CloserThan20minutes in \( T \) is updated as time passes, and therefore – at time \( t \) – it measures time from \( t \), and not from \( t' \). That is, even if at time \( t' \) the interval \( e' \) has been recognized as an instance of HavingMeal, the same interval would not be recognized any more at time \( t \), thus preventing \( e \) to be recognized as an instance of FollowingPrescription. This problem is avoided by making the assertion \( \text{HavingMeal}(e') \) explicit at time \( t' \).

5.2 Complexity Analysis

Since reasoning in \( \mathcal{EL}^{++} \) has a complexity which is polynomial with respect to the number of facts in the ontology, it is important to characterize how the number of assertions in \( \mathcal{A} \) increases as time passes. Remind that \( N \) is the number of sensors, and each sensor \( i \) can be in \( M_i \) different states. Since most assertions are made in \( \mathcal{A} \) when at least one sensor changes its state (Lines 9 to 41), we call \( T \) the number of times that there has been a change in sensor states from the beginning up to time \( t \), and characterize the number of assertions in \( \mathcal{A} \) as a function of \( T \).

Specifically, if the \( i^{th} \) sensor switches from state \( j \) to state \( l \), Lines 30 to 40 are executed if and only if the concept \( S_{ij} \) has been used to define a context or a situation in \( T \) and, similarly, Lines 10 to 29 are executed if and only if the concept \( S_{ij} \) has been used in \( T \). Executing Lines 30 to 40 at time \( t \) produce 1 concept assertion \( \text{Interval}(e_c) \), 1 role assertion \( \text{beginsAt}(e_c, t) \), 1 role assertion \( \text{hasPredecessor}(e_c, e_{c-1}) \), and a maximum of \( N \) role assertions \( \text{hasFluent}(e_c, s) \). This gives a number of new assertions which is \( O(N) \) in the worst case. Executing Lines 10 to 29 requires to clone a maximum of \( N - 1 \) individuals which already exist in \( \mathcal{A} \) together with all the corresponding role assertions but one. This gives a number of new assertions which is \( O(N^2) \) in the worst case, which happens when the sensor \( i \) is the one which has changed state less recently before \( t \), thus requiring to clone all individuals \( e \) in \( \mathbb{E} \) but one.

By considering Lines 9 to 41 all together, the overall number of assertions in \( \mathcal{A} \) as time passes turns out to be \( O(N^2T) \) in the worst case.
6 Experimental results

This Section presents experiments performed with data recorded in two different domains: the freely available dataset of the CASAS project \cite{61} by Washington University\(^4\), and the assisted living facility Villa Basilea in Genova, Italy.

6.1 CASAS dataset

The CASAS dataset contains sensor data recorded in many experiments in a Smart Home, during which 19 persons have been requested to perform different tasks. Sensors are mostly PIR sensors to detect presence, “item sensors” which specify the state of a given object, “door sensors” which specify if a door is closed or open, and finally a “phone sensor”, which specifies whether the receiver is picked up or not. To perform experiments, we randomly chose 2 out of the 19 persons, and we used the sensor data recorded during their operations to model the ontology (this is referred to as the modelling set). After the ontology has been built, the performance of the system are evaluated by trying to correctly classify the tasks performed by the remaining persons (this is referred to as the test set).

Figure 3 shows the OntoGraf of the PreparingMeal concept defined using the Protégé editor. Notice that, for visualization’s sake, the concept has been defined slightly differently from (19), since OntoGraf does not visualize descriptions but only concepts, i.e., named descriptions:

\[
\begin{align*}
\text{Microwave\_Open\_1} & \equiv \text{Microwave\_Open}\top \quad \exists \text{hasPredecessor.\ Pot\_Taken}, \\
\text{PreparingMeal} & \equiv \text{Microwave\_Closed}\top \quad \exists \text{hasPredecessor.\ Microwave\_Open\_1}.
\end{align*}
\]

Table 2 shows the results obtained. The rows correspond to different tasks (modelled as contexts or situations in the ontology), some of which have been described in Section 4. The second and third column report, respectively, the overall number of role restrictions and the number of hasPredecessor role restrictions which are required to define the corresponding situation. The fourth column reports the number of times that a situation has been correctly detected by the system, and the fifth column reports the number of times that the task has been actually performed. A success rate of 90.35\% has been reported (103 correctly classified situation out of 114). Failures

\(^4\)CASAS Smart Home Project, Washington State University - http://ailab.wsu.edu/casas/
are due to how contexts and situations are modelled, i.e., starting from the actions performed by the 2 initial users: these actions do not perfectly match the actions performed by the remaining 14 users when performing the same tasks\textsuperscript{5}.

Table 2: Situation recognition in a Smart Home

<table>
<thead>
<tr>
<th>Action</th>
<th>#roles</th>
<th>#hasPr.</th>
<th>Detected</th>
<th>Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakingPhonecall</td>
<td>3</td>
<td>2</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>CleaningHouse</td>
<td>6</td>
<td>3</td>
<td>14</td>
<td>19</td>
</tr>
<tr>
<td>PreparingMeal</td>
<td>4</td>
<td>2</td>
<td>17</td>
<td>19</td>
</tr>
<tr>
<td>HavingMeal</td>
<td>6</td>
<td>3</td>
<td>17</td>
<td>19</td>
</tr>
<tr>
<td>RefillingDispenser</td>
<td>3</td>
<td>1</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>WateringPlants</td>
<td>3</td>
<td>2</td>
<td>18</td>
<td>19</td>
</tr>
</tbody>
</table>

Some considerations must be done concerning the system performance. Complexity is \(O(N^2T)\) in the worst case, which is not acceptable if the system is designed to work continuously for a long time. For instance, in the CASAS dataset, the number of available sensors is about 70, among which we consider only \(N = 16\) sensors when defining contexts and situations. Figure 4 shows the behaviour of the system using Hermit [76], one of the fastest OWL reasoner nowadays available, during an experiment with the CASAS dataset. The upper plot reports the number of individuals in the ABox versus \(T\), whereas the lower plot reports the corresponding reasoning time on a i7 processor: after 568 iterations (corresponding to 30 minutes), the number

\textsuperscript{5}The software for reproducing these experiments is available at the address http://www.robotics.laboratorium.dist.unige.it.
of individuals is about 600, whereas reasoning time ranges from 20 to 50 seconds, which is not acceptable.

A solution has been implemented to significantly reduce the number of individuals and, consequently, reasoning time. Instead of keeping in the ABox all individuals of type Interval, we let the system remove them. Specifically, in Lines 26 and 45 of Algorithm 1, whenever an individual Interval(e) has been classified as being an instance of a concept D corresponding to a context or a situation, a concept assertion D(e) is explicitly added to the ABox, and made available for future classifications. Otherwise, the concept assertion Interval(e) and the corresponding role assertions can be simply removed after some time. Figure 5 on the top and Figure 5 in the middle report results about the same experiment in Figure 4, but with the additional mechanism to limit the number of individuals by removing them after 10 minutes: the performance of Hermit after 6000 iterations (corresponding to about 190 minutes) are significantly improved, with a reasoning time always lower than 2.5 seconds (155 ms on average). Figure 5 on the bottom reports results obtained with the JFact reasoner, a Java implementation of Fact++, which exhibits lower performance in this application (1.45 sec on average). See also [78] for a more general comparison of OWL 2 EL reasoners.

It should also be observed that, in this application domain, the worst case complexity $O(N^2T)$ is not very likely to be reached. Figure 6 shows an example, taken from the same experiment as above, of how the state of sen-

Figure 4: CASAS dataset. Top: number of individuals. Bottom: reasoning time with Hermit.
Figure 5: CASAS dataset. Top: number of individuals. Middle: reasoning time with Hermit. Bottom: reasoning time with JFact. Individuals of type Event which are not classified as contexts or situations are removed after 10 minutes.

Sensors changes as time passes: some sensors change their state very frequently (e.g., PIR sensors), whereas others change their state at a lower frequency. Sensors which change their state at a high frequency produce individuals of type Interval which are inserted and removed from $E$ before any other sensor changes its state, therefore requiring to add only a few individuals at every iteration of the algorithm (only one individual in the best case).

6.2 Assisted Living Facility Villa Basilea

The system has been implemented and is currently operating 24/7 within the assisted living facility Villa Basilea in Genova, Italy. The apartment has three bedrooms, each with two beds, a living room with a sofa, a bathroom, and a kitchen. Residents have meals outside the apartment: when at home, they mostly spend their time in the living room or, during night, in their beds. The purpose of the system is to detect anomalies in residents’ behaviours. A conservative approach is adopted: if an anomalous situation is detected, the system has the only responsibility of firing an alarm to raise the attention of professional caregivers through the visual interface shown in Figure 7. It is then up to them to further investigate, and to decide how to deal with the event occurred.
Figure 6: CASAS dataset. Plot of sensor states as they change in time.

Figure 7: Villa Basilea. Visual interface for caregivers (text is in Italian).
Sensors are distributed all over the apartment, and connected to the Smart Home system for real-time processing: they are PIR sensors (one for each bed, one in the bathroom, one on the sofa, one in the kitchen, one over the main entrance of the apartment), “door / window sensors” which return the state \{open, closed\} of doors and windows, and “light sensors” which return the state \{on, off\} of light switches. Differently from the CASAS dataset, we do not have “item sensor” which return the state of a given object, nor “phone sensors”. Then, it becomes of primary importance to exploit as much as possible the temporal relationships between lower-level sensor patterns in order to infer higher-level situations.

To define contexts, we start by introducing a set of virtual clock-based binary sensors, each corresponding to a period of the day (e.g., wake-up time, morning, lunch time, afternoon, evening, night). For instance, we introduce a “night sensor” whose state is \textit{Night} from 10pm to 8am (and \textit{NotNight} otherwise), and a “wake-up sensor” whose state is \textit{WakeUp} at 7.30am (and \textit{NotWakeUp} otherwise). Then, the following definitions

\[
\text{IsNight} \equiv \text{Interval} \cap \exists \text{hasFluent.Night}, \\
\text{IsWakeUp} \equiv \text{Interval} \cap \exists \text{hasFluent.WakeUp},
\]

(31)

clearly show that, at 7.30am, both contexts would be recognized in the same iteration of Algorithm 1.

PIR and door / window sensors are exploited as usual. For instance, the set of contexts

\[
\text{InBed} x \text{Room} y \equiv \text{Interval} \cap \\
\exists \text{hasFluent.PIR._Bed}.x \text{Room}.y \text{On},
\]

(32)

defined by performing all string substitutions \(x = 1, 2, y = 1, 2, 3\) corresponding to different beds and rooms, describe the fact that somebody is currently in bed \(x\) of room \(y\) (or close to it). The following additional contexts are self-explaining:

\[
\text{On._Sofa} \equiv \text{Interval} \cap \exists \text{hasFluent.PIR._Sofa}.\text{On}, \\
\text{Near._TV} \equiv \text{Interval} \cap \exists \text{hasFluent.PIR._TV}.\text{On}, \\
\text{InBathroom} \equiv \text{Interval} \cap \exists \text{hasFluent.PIR._Bathroom}.\text{On},
\]

\[
\text{Somebody._Awake._DiningRoom} \equiv \text{IsNight} \cap \\
\exists \text{hasFluent.Light._DiningRoom}.\text{On},
\]

(33)

\[
\text{Somebody._Awake._Bedroom} x \equiv \text{IsNight} \cap \\
\exists \text{hasFluent.Light._Room}.x \text{On}, \forall x, \\
\text{Somebody._Awake._Bedroom} x \subseteq \\
\text{Somebody._Awake._Bedroom}, \forall x.
\]
Situations are defined by exploiting the temporal relationships between
the contexts above. Among the others,

\[
\text{Resting} \equiv \text{On\_Sofa} \sqcap \exists \text{hasPredecessor}.(\text{On\_Sofa} \sqcap \\
\text{CloserThan2minutes} \sqcap \text{FartherThan30seconds}),
\]

(34)
describes a person resting on the sofa;

\[
\text{WatchingTV} \equiv \text{On\_Sofa} \sqcap \exists \text{hasPredecessor}.(\text{Near\_TV} \sqcap \\
\text{CloserThan2minutes}),
\]

(35)
describes a person watching the TV;

\[
\text{VisitingBathroomDuringNight} \equiv \exists \text{HasPredecessor}.(\text{IsNight} \sqcap \\
\text{InBathroom} \sqcap \text{FartherThan30seconds} \sqcap \\
\text{CloserThan2minutes}), \forall x, y,
\]

(36)
describes the case that somebody has raised from bed in room \(x\) to go to
the bathroom during night;

\[
\text{SomebodyDisturbing} \equiv \exists \text{HasPredecessor}.(\text{Somebody\_Awake\_DiningRoom} \sqcap \\
\text{InBed} \sqcap \text{FartherThan30seconds} \sqcap \\
\text{CloserThan2minutes}), \forall x, y,
\]

(37)
describes the case that somebody has raised from bed and turned on lights
both in the bedroom and the living room;

\[
\text{SleptInBed} \equiv \exists \text{HasPredecessor}.(\text{IsWakeUp} \sqcap \\
\text{InBed} \sqcap \text{FartherThan30seconds} \sqcap \\
\text{CloserThan2minutes}), \forall x, y,
\]

(38)
describes the case that a person has slept in bed \(x\) of room \(y\) during night;

\[
\text{SleptInBed\_Last3Days} \equiv \exists \text{HasPredecessor}.(\text{SleptInBed} \sqcap \\
\text{SleptInBed} \sqcap \text{SleptInBed} \sqcap \text{SleptInBed}), \forall x, y,
\]

(39)
describes the case that a person has slept in bed \(x\) of room \(y\) for 3 days
consecutively (this principle can be iterated for an arbitrary number of days).

Table 3 shows results acquire during three days of continuous monitor-
ing, by focusing on a subset of all the situations defined in the ontology (e.g.,
only results concerning bed 2 of room 1 are shown). The table reports the
overall number of role restrictions required to define the situation, the
number of hasPredecessor role restrictions, and finally the number of times that
each situation has been recognized. Notice that, in this case, it is difficult to
Table 3: Situation recognition in a Smart Home

<table>
<thead>
<tr>
<th>Role</th>
<th>#roles</th>
<th>#hasPr</th>
<th>Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resting</td>
<td>5</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>WatchingTV</td>
<td>4</td>
<td>1</td>
<td>197</td>
</tr>
<tr>
<td>VisitingBathroomDuringNight21</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Somebody_Disturbing</td>
<td>3</td>
<td>1</td>
<td>57</td>
</tr>
<tr>
<td>SleptInBed21</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>SleptInBed21_Last3Days</td>
<td>14</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

compare the outputs of the system with ground truth, since it would require a continuous monitoring of what is actually happening in the house. A random sampling performed by observing the actual scene through surveillance cameras returns an almost 100% success rate in the classification process: among the others, SleptInBed_xy has been correctly detected three times (once per night), and SleptInBed_xy_Last3Days has been correctly detected only once. Most failures concern the WatchingTV situation. This is due to how the situation has been modelled to get rid of the absence of a “TV sensor”: the designer has erroneously assumed that a person getting close to the TV, and later sitting on the sofa, is probably going to watch the TV, which is not always true (especially if nobody has lost the TV remote controller!).

By analysing Table 3 and inspecting the log returned by the system, it can be also observed that – under some circumstances – the same situation is recognized multiple times in subsequent iterations of the algorithm. For instance, if a person is on the sofa, the PIR sensor switches on and off a number of times, and this ultimately affects the recognition of contexts and situations which depend on that sensor. However, from the point of view of the caregiver which monitors the residents’ activity through the visual interface provided by the system, this is not a problem: if the situation is recognized multiple times, the visual interface simply produces a flashing alert. A low-pass filter can be easily implemented to avoid this intermittent behaviour, however it is not discussed here.

Figure 8 on the top reports the number of individuals as it varies during three days (starting from 4pm of day one to 1pm of day three), and Figure 8 on the bottom reports the time required for reasoning (355 ms on average), validating the results obtained with the CASAS dataset.
Figure 8: Villa Basilea. Top: number of individuals. Bottom: reasoning time with Hermit.

7 Conclusions

The present article has described a system for describing and recognizing patterns of events in a Smart Environment which is based on ontologies expressed in the $\mathcal{EL}^{++}$ Description Logics formalism. Differently from other systems proposed in the literature, our work do not rely on any external reasoning mechanism or language to recognize the occurrence of relevant events: instead, it relies on the basic mechanisms for reasoning in ontologies (i.e., subsumption and instance checking), even to detect patterns of events which are temporally related to each other.

The system has been implemented in OWL 2 EL; the complexity analysis shows that reasoning works in polynomial time, therefore making the approach implementable in real-world scenarios where the ontology is populated starting from sensor data recorded over time.

Experiments performed in two Smart Home domains have returned promising results in terms of ease of design, recognition rate and run-time performance.

References


