Situation Awareness Meets Ontologies: A Context Spaces Case Study

Andrey Boytsov[1(✉)], Arkady Zaslavsky[2], Elif Eryilmaz[1], and Sahin Albayrak[1]

[1] Distributed Artificial Intelligence Laboratory (DAI-Labor),
Technical University of Berlin (TU-Berlin), 10587 Berlin, Germany
{andrey.boytsov,elif.eryilmaz,sahin.albayrak}@dai-labor.de
[2] Digital Productivity Flagship,
Commonwealth Scientific and Industrial Research Organization (CSIRO),
Melbourne, Australia
arkady.zaslavsky@csiro.au

Abstract. Efficiency and appeal of pervasive computing systems strongly depends on how well and robustly they represent and reason about context and situations. Populating situation search space and inferring situations from context which, in turn, is computed from fusing sensor data and observations remains a major research challenge. This paper proposes to use ontologies as representation of domain knowledge to generate situation search space and then match context with already defined situations. To illustrate the feasibility, a context spaces approach is used to represent, generate and reason about situations as abstractions in a multidimensional space. The proposed approach is evaluated and discussed.

1 Introduction

Efficiency and appeal of pervasive computing systems strongly depends on how well and robustly they represent and reason about context and situations. Populating situation search space and inferring situations from context which, in turn, is computed from fusing sensor data and observations remains a major research challenge. In this paper we propose a novel approach to build fast and intuitive approach for context awareness, situation awareness and sensor fusion in pervasive computing system.

Bazire and Brézillon [4] analyzed a set of 150 definitions of context for different subject areas. The authors noted that there is no consensus about some aspects of context and there is no single definition. However, common understanding is that "*context is the set of circumstances that frames an event or an object*" [4] and "*context acts like a set of constraints that influence the behavior of a system (a user or a computer) embedded in a given task*" [4]. It complies with the most popular definition of context in pervasive computing, which was proposed by Dey and Abowd [11]. They defined context as "*any information that can be used to characterize situation of an entity*" [11]. It should be noted that this definition was one of the definitions analyzed by Bazire and Brézillon [4]. The definition is very general and yet it provides sufficient criteria to identify context-related information among sensor data and user input. Location, identity, activity and time are the main aspects of context. It should be also specifically noted that the word *situation* in the

H. Christiansen et al. (Eds.): CONTEXT 2015, LNAI 9405, pp. 3–17, 2015.
DOI: 10.1007/978-3-319-25591-0_1

definition by Dey and Abowd means *circumstances* or *conditions*, and does not imply rigorous definition of situation in pervasive computing, which will be presented later in this section. Pervasive system is context aware "*if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task*" [11]. Context awareness is a key feature of any pervasive computing system.

Context awareness can be enhanced by appropriate situation awareness. Throughout the paper we are going to use the definition of situation proposed by Ye et al. [19]. From pervasive computing perspective Ye et al. [19] define a situation as "*external semantic interpretation* of sensor data". Like a definition of context, this is a very general definition, yet it provides criteria for technical implementation of situation recognition. *Semantic interpretation* means that "situation assigns meaning to sensor data" and *external* implies "from the perspective of applications" [19]. Therefore, situation recognition can be viewed as one of the final stages of context awareness – extracting the essential meaning of context. For example, context processing results like "user is sitting" or "presence in the living room" can be viewed as situations.

The algorithm of *semantic interpretation* of sensor data takes sensor values as input and provides the interpretations as output. In the output, each possible situation is assigned a value, which can be Boolean value (whether a situation is occurring), or probability of a situation occurrence, or fuzzy confidence level. Sometimes reasoning algorithm can be formulated as a single mathematical expression, and this expression will be referred to as *situation formula* in this paper.

Ontologies are domain-independent way of storing semantic information [14]. There are multiple ontologies for sensor networks [3, 8, 9] and for pervasive computing context [2, 18, 19]. Context ontologies often include situation awareness aspects.

In this paper we propose a novel approach and a set of algorithms that generate situation formulas. Those formulas enable fast and lightweight situation awareness in pervasive computing system. As an input the proposed approach uses pervasive computing system description in terms of context and sensor ontologies. As a part of situation generation approach we propose a method that bridges the gap between context and sensors and introduces an integrated ontology of a pervasive computing system.

The rest of the paper is structured as follows. Section 2 contains related work overview. Section 2 also identifies the scope of work to obtain situation formulas out of context and sensor ontologies. Section 3 proposes the main principles of situation formulas generation. Section 4 further elaborates on those principles and proposes the detailed situation formula generator algorithm. Section 5 introduces context spaces approach, which is essential for demonstrating generated situation formulas. Section 6 finalizes generation of situation formulas out of integrated ontologies. Section 7 provides discussion and evaluation of the proposed approach. Section 8 proposes future work directions and concludes the paper.

2 Related Work

Ontology in artificial intelligence can be defined as "*explicit specifications of a conceptualization*" [14]. Ontologies provide a domain-independent way for

knowledge representation, sharing and reasoning [14]. Multiple surveys [2, 18, 19] provide overviews of ontology-based context reasoning approaches.

SOUPA [7] (Standard Ontology for Ubiquitous and Pervasive Applications) is one of the earliest examples of ontologies in pervasive computing. SOUPA was based on CoBrA-ONT [6] (ontology for Context Broker Architecture). SOUPA already had the vocabulary to describe *situational conditions* like "in a meeting" or "out of town". Situational conditions were actually situations by the definition of Ye et al. [19].

Dapoigny and Barlatier [10] used ontological modelling to overcome the lack of expensiveness in logic-based theories. They formalized the context by using the Calculus of Inductive Constructions in the lower layer and an ontological layer in the upper layer.

Wang et al. [17] developed CONON (CONtext ONtology) ontology. CONON used the concept of context entities, which is a general term for concepts like location, person or activity. Situations were defined in terms of rules, like in formula (1).

$$\begin{aligned}&(?u\ \text{locatedIn Bedroom})\ \&\ (\text{Bedroom lightLevel LOW})\ \&\\ &(\text{Bedroom drapeStatus CLOSED})\ \rightarrow\ (?u\ \text{situation SLEEPING})\end{aligned} \tag{1}$$

Gu et al. [15] introduced SOCAM system (Service Oriented Context Aware Middleware). Along with other features SOCAM contained context ontology, which was based on CONON. The concept of situation in SOCAM is similar to CONON.

Anagnostopoulos et al. [1] proposed situation awareness technique based on combination of ontologies and fuzzy logic. Situations were represented as concepts, and the relation "person is involved in a situation" (that effectively means "situation is occurring for this person") was reasoned about using fuzzy logic (see formula (2)).

$$\bigwedge_{i=1}^{N} context\left(x_i, user\right) \rightarrow IsInvolvedIn\left(Situation, user\right), N > 1 \tag{2}$$

Formula (2) is quite similar to formula (1) – situations are reasoned about in terms of rules. Anagnostopoulos et al. [1] also introduced the relation "disjoint", which means that situations cannot occur at the same time (like "Meeting" and "Jogging").

Ejigu et al. [13] proposed ontology-based generic context management model (GCoMM). GCoMM mainly relied on rules and queries for inference and decision making. Although the concept of situation was not explicitly introduced, some rules could be viewed as situation inference rules. For example, consider formula (3) [13].

$$\begin{aligned}&[\text{rule1: ?user1 nsp:locatedIn ?roomN})\ \&\ (?\text{user2 nsp:locatedIn ?roomN})\\ &\qquad -> (?\text{user1 nsp:coLocatedWith ?user2})]\end{aligned} \tag{3}$$

According to the related work, there are two available concepts of situations:

1. Potential situation is an entity. The relation connects context entity to potential situation, if that situation is occurring. This approach is applicable if a situation refers to a single context entity like "*User* is sleeping" or "Presence in the *Living Room*". Formulas (1) and (2) provide an example. In formula (1) the relation is called

situation [17], while in formula (2) the relation is called *IsInvolvedIn* [1]. However, the meaning of both relations is similar.

2. Potential situation is a relation that connects context entities. The relation exists if the situation is occurring. This approach is applicable if a situation refers to two context entities at once. For example, "*user:Alice* coLocatedWith *user:Bob*" or "*user:Alice* locatedIn *location:LivingRoom*" (formulas (2) and (3)). Relationship *locatedIn* and *nsp:coLocatedWith* are situations according to the definition, i.e. "external semantic interpretation of sensor data" [19].

Note that context ontology sometimes combines both approaches, and the chosen approach for any particular situation is determined by whether the situation is parametrized by one or two context entities. For example, formula (1) combines situation *Sleeping*, defined as an entity, and situation *locatedIn*, defined as a relation. Therefore, situation formula generator should be able to work with both approaches at once.

The common aspect of both approaches is that situation reasoning is performed in terms of rules. This common aspect is an important factor that allows situation formula generation using both situation representation approaches.

Usually context ontologies do not contain information about the sensors. The facts like "Bedroom lightLevel LOW" (see formula (1)) are assumed to be already inferred from the sensed information. Context ontology does not contain information about reasoning in case there are multiple sensors that measure light level or in case there are none. Therefore, context ontology should be augmented with information about sensors in order to generate situation formulas, which take sensor values as input.

There are some research efforts to construct ontology-based sensor description and data modelling [8, 9]. The challenge is in the generic representation of sensor data and characteristics to have a common level of sensor representation. The main approach how to capture the critical characteristics of a sensor accurately is the metadata annotation for sensor characteristics [3]. Although there are many efforts on defining sensor meta-information, the description of observations measured by sensors has not been addressed much. The W3C Incubator Group released Semantic Sensor Network XG Final Report, which defines SSN ontology [21] allowing describing sensors, including their characteristics. SSN ontology focuses on providing a domain independent ontology which is generic enough to adapt to different use-cases and compatible with the OGC standards [22] at the sensor and observation levels.

To summarize the related work, situations in most context ontologies are represented in terms of parametrized rules. Therefore, situation generator should accept parametrized rules as input. Moreover, context ontologies usually do not contain sensor information. Therefore, situation formula generator should accept both context and sensor ontology as input. Next section takes into account the requirements and proposes general approach to situation formula generation.

3 Fusion of Context and Sensor Ontologies

Previous section identified the input format for situation generator. The input includes sensor and context ontologies, which includes situations as parametrized rules.

Context and sensor ontologies contain much common knowledge that does not depend on exact pervasive computing system. For example, CONON [17] contains information about context entities (like concepts of location, person or activity) and their subclasses. In this article we use CONON ontology as an example, but proposed principles and algorithms can be applied to other context ontologies.

Situation in formula (1) is applicable to any smart home. The only requirement is that parts of the situation like "light level in bedroom is low" and "user is in bedroom" can be inferred from sensor data. For any particular pervasive computing system only the instances of classes should be added: what kind of rooms are there in the smart home (instances of *location*), who lives there (instances of *person* class), etc.

In turn, SSN sensor ontology [21] mainly consists of general information about sensors, the values they measure and measurement capabilities of the sensors. The only thing left to add for particular pervasive computing system (like smart home) is the exact sensors we have and where they are.

The approach to situation formula generation can be summarized in Fig. 1.

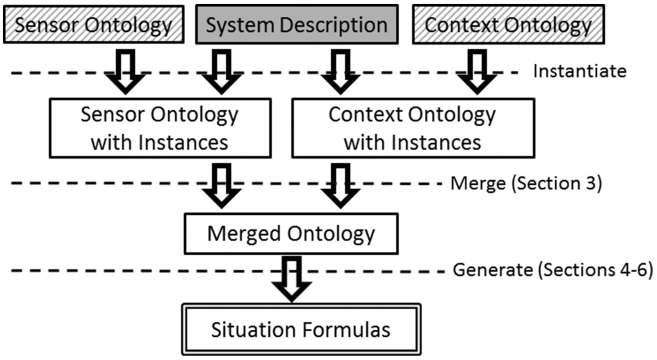


Fig. 1. High-level system description for situation formulas generator. Pervasive system description is an input. Context and sensor ontologies are static input information, which does not depend on the system under analysis. Situation formulas are the output of the approach.

Context and sensor ontologies can be integrated as follows. Each sensor can be viewed as either measuring user characteristic (e.g. mobile phone measuring user's location and acceleration) or place characteristic (e.g. light level or motion sensor in the room). Location and person are context entities in CONON context ontology. Therefore, we need a relation that specifies that some instance of sensor belongs to some context entity. We define relation *belongsTo*, which connects hardware platform with context entity (Fig. 2). As will be shown in Sects. 4–6, this relation is enough to integrate context and sensor ontology for situation generation purpose.

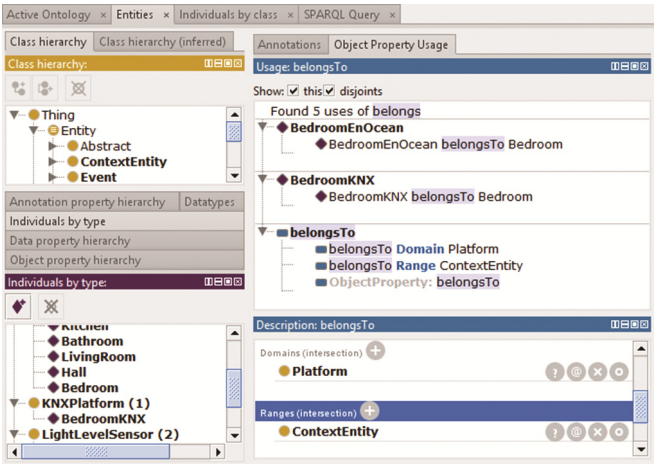


Fig. 2. Proposed connection between SSN ontology [21] and context ontology. The example was implemented in PROTÉGÉ [23] tool and shows the definition of new *belongsTo* relation.

This section proposed general approach to situation generation and described the steps to integrate context and sensor ontology. Next section further elaborates that approach and provides the exact algorithm for situation generation, i.e. (in terms of Fig. 1) details for transition from integrated ontology to exact situation formulas.

4 Generation of Situation Formulas

As an example context, we assume that users Alice & Bob live in the smart home, and the smart home has locations living room, bedroom, bathroom and hall. Situations from expressions (1) and (3) are used as running examples. The example was implemented in PROTÉGÉ [23] tool using the combination of SSN [21] ontology and elements of CONON ontology, recreated using the description in the paper [17].

The first step involves recognizing the dependencies between situations. For example, in formula (1) the situation "User is sleeping" is defined as conjunction of conditions "User located in bedroom", "bedroom light level low" and "bedroom drape status closed". The condition "User located in bedroom" (for each particular user) is a situation on its own, and its inference differs a lot depending on what kind of indoor localization systems are used. Therefore, situation "User is sleeping" depends on situation "User located in bedroom" Those dependencies can be summarized in the following directed graph (Fig. 3).

Dependency graph can be obtained by relatively straightforward approach - connection is drawn whenever the situation rule mentions another situation. We assume that the dependency graph is acyclic.

Situation formulas should be generated in the order defined by dependency graph: each situation is processed only after all its dependencies are processed. The assumption that the graph is acyclic ensures that such order of processing should exist. Therefore, for the running example situation "User located in" should be generated first, and then

situations "User is sleeping" and "User is co-located with another user" can be generated in any order. For the example we take only formulas (1) and (3), we assume that situation "User located in" is already generated. A formula for a single situation can be generated using the algorithm below (see the pseudocode).

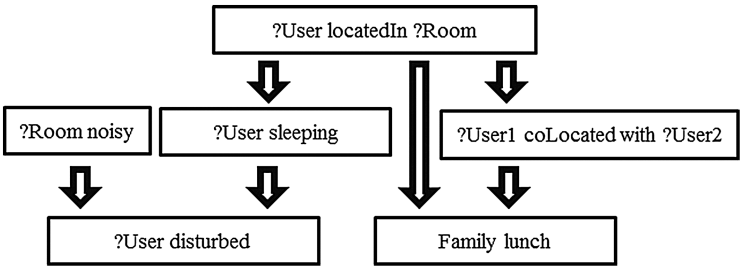


Fig. 3. Dependencies between situations. Apart from situations from formulas (1) and (3), it contains several more situations. This figure is designed for illustration purpose, and does not account for reasoning uncertainty that inevitably occurs in real-life scenarios.

Main situation generator algorithm – pseudocode

```
generateSituations(rulesList)
   results = new List<ContextSpacesSituation>();
   forEachCombination
         parameterCombination:rulesList.getParameters()
      rulesListSituation = rulesListToSituation
           (rulesList, parameterCombination)
      results.add(rulesListSituation);
      addToSituationCache(rulesListSituation);
  end forEachCombination
  return results;
```

To summarize the pseudocode, the first task for situation generator is unrolling the parameters. For situation "User is sleeping" there is only one parameter – the person. Therefore, there will be two situations generated "Alice is sleeping" and "Bob is sleeping". For the second example there will be formally four situations generated – "Alice co-located with Alice", "Alice co-located with Bob", "Bob co-located with Alice", and "Bob co-located with Bob". Detection of symmetric situations ("Alice co-located with Bob" vs "Bob co-located with Alice") and always true situations ("Alice co-located with Alice") is a part of future work.

The pseudocode above manages situation dependencies using the cache. Situation "Bob is in bedroom" is generated before "Bob is sleeping" (see Fig. 3) and put into the cache (*addToSituationCache*). Later, when generating "Bob is sleeping", the situation "Bob is in bedroom" is retrieved from cache and used as a condition.

The function *rulesListToSituation* will be called for each of the situations mentioned above. Note that there might be several rules for the same situation, corresponding to multiple ways to infer the same situation. As will be shown further, sometimes the presence of multiple rules is implicit.

Generator algorithm for instantiated situations – pseudocode

```
rulesListToSituation (rulesList, parameterCombination)
   disjuncts = new List<Situation>();
   foreach rule : allSituationRules
       disjuncts.add(ruleToSituationFormula
            (rule, parameterCombination));
   end foreach
  return OR(disjuncts);
```

The pseudocode above means that if there are several rules for the same situation, they are combined using OR logical operator. For situation "user is sleeping" there is a single rule in formula (1). However, for "co-located" situation in formula (2) several rules are implied. There is a parameter *?roomN*, which is mentioned in the left side of the situation rule, but it is not mentioned in the right side. Therefore, it implicitly creates multiple rules, which should be combined using OR operator. For example, situation "Alice co-located with Bob" can be decomposed as "Alice co-located with Bob in the kitchen" OR "Alice is co-located with Bob in the living room" OR "Alice co-located with Bob in the hall" etc. At least one of those subsituations should be true in order for "Alice co-located with Bob" to be true. Therefore, they should be combined using logical OR.

Next pseudocode introduces situation generation from a single rule.

Transformation algorithm for a single rule – pseudocode

```
ruleToSituationFormula(rule, parameters)
  conjuncts = new List<Situation>();
  foreach condition : rule.getConditions()
     if (situationInCache(conditon, parameters))
        conjuncts.add(situationFromCache
              (conditon, parameters));
     else
        situationForCondition = conditionToSituation
              (conditon, parameters)
        conjuncts.add(situationForCondition);
        addToSituationCache(situationForCondition);
     end if
    end foreach
    return AND(conjuncts);
```

The pseudocode above combines all the conditions using AND operator. It follows the rules quite straightforwardly. However, the generated situations for each condition are cached and then retrieved from the cache if necessary. Note that for situations like "User is sleeping" there are several parameter-independent parts. In this example those are conditions "Bedroom light level low" and "Bedroom drape status on". Once calculated for a single user, there is no need to recalculate them for another user. So, the situation "Bedroom light level low", generated as a part of "Alice is sleeping" situation will be stored in the cache and later reused as a part of "Bob is sleeping" situation.

Some conditions are represented as situations. Those conditions are already stored in the situation cache as a part of situation dependency management (see earlier in this section). Therefore, those situations will be retrieved from the cache and reused.

The function *conditionToSituation* needs to process the concepts like "light level low". It also needs to handle the case like, for example, several light level sensors in the

bedroom. Before introducing the pseudocode for *conditionToSituation*, we need to introduce context spaces approach.

5 Context Spaces Approach

There are following requirements to situation reasoning approach:

- The approach should support fuzzy concepts. This requirement follows from the presence of rule conditions like "light level low" (formula(1)).
- The approach should support logical operators like AND and OR. It follows from the pseudocode for *ruleListToSituation* and *ruleToSituationFormula*.
- The approach should allow combining the data from multiple sensors. It follows from considerations discussed in Sect. 3.

For this purpose we propose to use context spaces approach [16]. In order to fully incorporate fuzzy concepts, we are going to use an extension of context spaces approach proposed by Delir et al. [12]. Context spaces approach represents all possible contexts as a multidimensional space, which is referred to as *context space*.

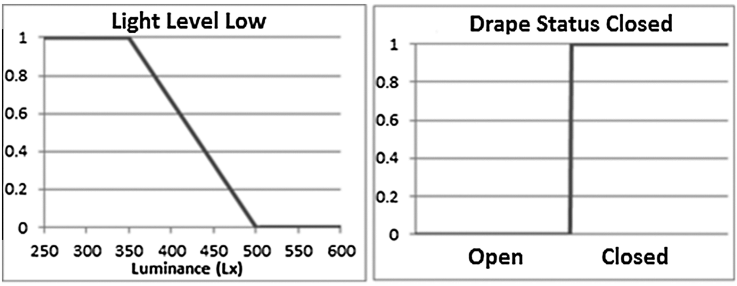
A domain of values of interest (i.e. an axis in *context space*) is referred to as a *context attribute*. For example, air temperature, light level, air humidity can be relevant context attributes in a smart home. Non-numeric values, like on/off switch position or open/closed drape status, can also be context attributes. In our approach every sensor will have its own corresponding context attribute, and sensor fusion will be done on situation awareness level.

The situations are represented using *situation spaces*, which can be viewed roughly as subspaces of multidimensional context space. The output of situation reasoning is confidence level, which falls within [0;1] range. A situation confidence value in context spaces approach is viewed as a combination of contributions of multiple context attributes. Confidence level can be determined using formula (4).

$$conf_S(X) = \sum_{i=1}^{N} w_i * contr_{S,i}(x_i) \tag{4}$$

The term $conf_S(X)$ represents confidence level for a situation S. The context X contains the context attribute values x_i, and each of those context attributes are assigned the importance weights w_i (all the weights sum up to 1). The contribution function of i-th context attribute into situation S is defined as $contr_{S,i}(x_i)$. Figure 4 presents plausible contribution functions for "User is sleeping" situation.

Note that rule-based situations are represented not as single context space situations, but as logical combinations of context spaces situations. For that kind of reasoning the context spaces approach contains the concept of situation algebra. Formula (5) provides the definition of basic operators of situation algebra. The definitions are compliant with Zadeh operators [20].



(a) (b)

Fig. 4. Plausible fuzzy concepts for "User is Sleeping" situation. (a) Confidence level for "Light Level Low" condition depending on light level, (b) Confidence level for "Drape Status Closed" condition depending on drape status.

$$\begin{aligned} &AND{:}(A\,\&\,B)(X) = min(A\,(X)\,, B\,(X)) \\ &OR{:}(A\,|\,B)(X) = max(A(X), B(X)) \\ &NOT{:}(\neg A)(X) = 1 - A(X) \end{aligned} \tag{5}$$

As a result, context spaces approach complies with the requirements that are summarized in the beginning of this section. Next section finalizes situation generation algorithm and finished extraction of situation formulas from context and sensor ontologies. The extracted formulas are compliant with fuzzy context spaces approach [12].

6 Transformation of Rule Condition

Sections 3 and 4 provided the general approach to situation formula generator, and there was only one component missing: transformation of rule condition into the situation formula for runtime reasoning. That method was referred to as *conditionToSituation* in the pseudocode. This section finalizes *conditionToSituation* method and concludes the situation generator algorithm.

SPARQL [24] is a query language for ontologies. All the sensors that belong to certain context entity can be retrieved by the following SPARQL query.

Sensor list query - SPARQL

```
SELECT ?sensor WHERE {
 ?sensor :onPlatform ?platform .
 ?platform :belongsTo <ContextEntity> .
 ?sensor rdf:type ?sensorClass .
 ?sensorClass rdfs:subClassOf* ?widerSensorClass .
 ?widerSensorClass owl:onProperty :observes .
 ?widerSensorClass owl:hasValue <MeasuredProperty>
}
```

The SPARQL query effectively asks "Which sensors measure < *MeasuredProperty* > for a certain < *ContextEntity* >*?*". The query has two parameters. For example, for the condition "Bedroom lightLevel LOW" the parameter *MeasuredProperty* is

substituted by *Luminance*. We assume that there is a map, which connects the name of relation in context ontology (like *lightLevel*) to the name of characteristic, measured by sensor (like *Luminance*). That map can be implicitly encoded into ontology itself by introducing some additional connection between context and sensor ontology.

ContextEntity parameter is replaced by the exact context entity of interest. For example, for condition like "Bedroom lightLevel LOW" the context entity of interest is *Bedroom*, the instance of location class.

The results of the SPARQL query are used by the following pseudocode.

Transformation algorithm for a single condition - pseudocode

```
conditionToSituation(condition, parameterCombination)
   result = new ContextSpacesSituation();
   instantiatedCondition =
          condition.instantiate(parameterCombination);
   sensorList = SPARQLquery(
          instantiatedCondition.getContextEntity(),
          instantiatedCondition.getProperty());
   if (sensorList.size()==0)
    fail 'Property not measured`
   end if
   weight = 1/sensorList.size();
   foreach attribute : sensorList.attributes()
      result.addContextAttribute(attribute, weight,
          sensorList.getFuzzyFunction(attribute));
   end foreach
   return result
```

So, the pseudocode above generates a subsituation for each condition. At first, the method *SPARQLquery* requests the list of sensors that measure certain characteristic of a certain context entity. Sensor readings correspond to certain context attributes in context space. For example, for situation "User is Sleeping" it can retrieve 2 sensors that measure light level in the room. Let's say, those are *BedroomLightSensor1* and *BedroomLightSensor2*. We assume that sensor names correspond to the names of context attributes, i.e. axes in multidimensional context space. Each of those sensors is taken with equal weight, and each context attribute is given the same contribution function. In that case it will be "Light level low" contribution function from Fig. 4(a). To summarize, condition "*Bedroom light level LOW*" will be transformed to the following situation (formula 6). Other conditions will be processed in the similar manner according to the same algorithm.

$$\begin{aligned} &Conf\,(BedroomLightLevelLow) = \\ &\quad = 0.5 * contr\,(BedroomSensor1) + 0.5 * contr\,(BedroomSensor2) \\ &contr\,(x) = \begin{cases} 1, & x < 350Lx \\ \frac{500-x}{150}, & x \in [350, 500]Lx \\ 0, & x > 500Lx \end{cases} \end{aligned} \tag{6}$$

This concludes the situation formula generator algorithm. Situation formula can be generated from ontologies using a set of methods described in Sects. 3–6. Next section will provide evaluation and discussion of the proposed approach.

7 Evaluation and Discussion

This article proposes a novel approach to situation awareness in pervasive computing systems. The proposed approach contains two main steps. The first step, integrating the ontologies, connects context aspects to the methods of measuring the context features. The second step, transforms the representation of situations from the format of ontology into the format of context space. The algorithms were designed to maintain the equivalence of situation representations, i.e. the same sensor readings should result in the same reasoning outcomes, whether the situations are represented as ontology or as context spaces. Context space representation provides additional synergy and introduces the following benefits.

Identifying Essential Information. The ontology aims to capture all available information about pervasive computing system. This information gives a lot of insights for many aspects of pervasive system, including situation awareness. However, for situation awareness at the runtime only small part of that information is relevant. The proposed approach preprocesses the ontology information, singles out the information necessary for situation reasoning and leaves the rest out of reasoning process.

Efficient Lightweight Reasoning. The complexity of situation reasoning is summarized in Table 1. Complexity estimations are based on formula (4), where each contribution function is a piecewise linear function like in Fig. 4. So, the total complexity is of order $O(\sum_{i=1}^{M} N_i)$, where N_i is the number of context attributes involved in i-th situation and M is the number of situations in the context space. It should be noted that M can be larger than the number of situations in ontological representation. It can happen due to the unfolding of parameters ("User is sleeping" corresponds to "Alice is sleeping" and "Bob is sleeping") and due to the fact that each ontological rule is represented as a logical formula over situations in the context space (like "Bob is sleeping" is represented as AND-operation over 3 small situations – "Bob is located in the bedroom", "Draping status - closed" and "Light level in Bedroom is low").

Sensor Fusion. Situation rules in context ontologies do not specify what to do if there are multiple ways to measure the context feature, which influences situation reasoning (e.g. what if there are multiple light sensors for situation "user is sleeping"). Generation of context space situations allows fusing the values of multiple sensors. Smart sensor fusion is one of the directions of future work.

Reasoning Under Uncertainty. Proposed situation reasoning approach is robust to uncertainty, both in terms of measurement uncertainty (which is partially covered by previous paragraph) and uncertainty in terms of concepts (like "Light level Low").

Focusing on Most Relevant Situations. With growing amount of entities in the context (like users, locations, etc.), the number of possible situations can grow polynomially. Context spaces allow explicitly selecting situations of interest and, therefore, saving computational power by reasoning only about relevant situations.

Table 1. Complexity of situation reasoning – single situation

Operations	Order	Explanation
Additions	O(N)	The sum in formula (4) contains N summands. Each calculation of contribution function contains one more addition at most.
Multiplications	O(N)	Formula (4) contains N multiplications – one in every summand. Contribution function contains at most one more multiplication.
Division	None	The formulas for each linear fragment of contribution function can be precompiled in the form y = k*x + b. It eliminates the need for any divisions.
Comparisons	O(N)	There are N contribution functions, each calculation requires at most 4 comparisons – interval of piecewise function needs to be determined (see Fig. 4).
Time	O(N)	Complexity for each relevant operation grows linearly with the number of involved contribution functions or subsituations.

Generation of context spaces situations also introduces some challenges:

Preprocessing. Preprocessing can be done only once for any particular pervasive system. However, situations should be generated again if context model or sensors changed significantly (e.g. new sensor, new user, or user moving out).

Reasoning Explanation. Situation reasoning rules provide very intuitive representation of situations. Moreover, if the situation is triggered, it is relatively easy to identify and present to the user the factors that lead the pervasive system to assume that the situation is occurring. Context spaces provide intuitive representation of the situations. However, generation of explanations (like "Why did the system decide that it is a family lunch?") is slightly more complicated comparing to ontologies.

Next section identifies the direction of future work and concludes the paper.

8 Conclusion and Future Work

In this paper we proposed and developed fast, lightweight and robust approach to context awareness, situation awareness and sensor fusion in pervasive computing systems. As an input, the information about different aspects of pervasive computing systems was represented by two separate ontologies – context ontology and sensor ontology. We proposed and proved a method to integrate those ontologies, and achieve unified representation of complete pervasive computing system. On top of the unified representation we developed a method to extract all the necessary information for runtime situation

awareness and generate lightweight and intuitively clear situation formulas, which allows handling sensor fusion, reasoning under uncertainty, and working with fuzzy concepts.

We identified the following main directions of the future work:

- **Generation of Additional Situations**. From the rule (1) we generated formulas for situations like "Alice is sleeping" and "Bob is sleeping". It is also possible generate situation "Someone is sleeping", i.e. leave the parameters undefined. That situation might require different set of sensors (for example, it can be inferred using motion sensors without any involvement from indoor localization system).
- **Smart Sensor Fusion**. At the moment readings from several sensors are combined using equal weights. However, sometimes other approaches are preferred. For example, movement is detected if at least one motion sensor is triggered. Or if there is severe disagreement between sensors, pervasive system might consider discarding some sensor values as unreliable.
- **Using Derived Measurement**. Some context characteristics can be obtained even if they are not directly measured. For example, absolute acceleration might be derived using relative acceleration and orientation sensors, which both reside on user's mobile phone. Those derived values can be introduced by extending SSN ontology with artificial virtual platforms that will correspond to derived values.
- **Handling Disjoint Situations.** Situations like "User is standing" and "User is sitting" should not co-occur together. Some context ontologies [1] allow declaring situations as disjoint. In order to generate situations as disjoint, situation verification [5] can be extended and used it in conjunction with situation formula generation.

References

1. Anagnostopoulos, C., Ntarladimas, Y., Hadjiefthymiades, S.: Situational computing: an innovative architecture with imprecise reasoning. J. Syst. Softw. **80**(12), 1993–2014 (2007)
2. Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D.: A survey of context modelling and reasoning techniques. Pervasive Mob. Comput. **6**(2), 161–180 (2010)
3. Barnaghi, P., Meissner, S., Presser, M., Moessner, K.: Sense and sens' ability: semantic data modelling for sensor networks (2009)
4. Bazire, M., Brézillon, P.: Understanding context before using it. In: Leake, D.B., Kokinov, B., Dey, A.K., Turner, R. (eds.) CONTEXT 2005. LNCS (LNAI), vol. 3554, pp. 29–40. Springer, Heidelberg (2005). doi:10.1007/11508373_3
5. Boytsov, A., Zaslavsky, A.: Formal verification of context and situation models in pervasive computing. Pervasive Mob. Comput. **9**(1), 98–117 (2013)
6. Chen, H., Finin, T., Joshi, A.: An ontology for context-aware pervasive computing environments. Spec. Issue Ontol. Distrib. Syst. Knowl. Eng. Rev. **18**, 197–207 (2003)
7. Chen, H., Finin, T., Joshi, A.: The SOUPA ontology for pervasive computing. In: Tamma, V., Cranefield, S., Finin, T. (eds.) Ontologies for Agents: Theory and Experiences. Whitestein Series in Software Agent Technologies, pp. 233–258. Springer, Switzerland (2005)
8. Compton, M., Henson, C.A., Lefort, L., et al.: A survey of the semantic specification of sensors. In: CEUR Workshop Proceedings, October 2009

9. Calbimonte, J.P., Yan, Z., Jeung, H., et al.: Deriving semantic sensor metadata from raw measurements. In: 5th International Workshop on Semantic Sensor Networks, in conjunction with the 11th International Semantic Web Conference (ISWC), November 2012
10. Dapoigny, R., Barlatier, P.: Formalizing context for domain ontologies in Coq. In: Brézillon, P., Gonzalez, A.J. (eds.) Context in Computing, pp. 437–454. Springer, New York (2014)
11. Dey, A.K., Abowd, G.D.: Towards a better understanding of context and context-awareness. In: CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness, pp. 304–307 (2000)
12. Delir Haghighi, P., Krishnaswamy, S., Zaslavsky, A., Gaber, M.M.: Reasoning about context in uncertain pervasive computing environments. In: Tröster, G., Lombriser, C., Kortuem, G., Havinga, P., Roggen, D. (eds.) EuroSSC 2008. LNCS, vol. 5279, pp. 112–125. Springer, Heidelberg (2008)
13. Ejigu, D., Scuturici, M., Brunie, L.: An ontology-based approach to context modeling and reasoning in pervasive computing. In: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops, IEEE Computer Society, pp. 14–19 (2007)
14. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (1993)
15. Gu, T., Pung, H.K., Zhang, D.Q.: A service-oriented middleware for building context-aware services. J. Netw. Comput. Appl. **28**(1), 1–18 (2005)
16. Padovitz, A., Loke, S.W., Zaslavsky, A., Burg, B.: Verification of uncertain context based on a theory of context spaces. Int. J. Pervasive Comput. Commun. **3**(1), 30–56 (2007)
17. Wang, X.H., Zhang, D.Q., Gu, T., Pung, H.K.: Ontology based context modeling and reasoning using OWL. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, pp. 18–22 (2004)
18. Ye, J., Coyle, L., Dobson, S., Nixon, P.: Ontology-based models in pervasive computing systems. Knowl. Eng. Rev. **22**(4), 315–347 (2007)
19. Ye, J., Dobson, S., McKeever, S.: Situation identification techniques in pervasive computing: a review. Pervasive Mob. Comput. **8**(1), 36–66 (2012)
20. Zadeh, L.A.: Fuzzy sets. Inf. Control **8**, 338–353 (1965)
21. w3.org: Semantic sensor network XG final report: w3c incubator group report. http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/, June 2011. Accessed 14 Aug 2015
22. OGC observations and measurements: http://www.opengeospatial.org/standards/om. Accessed 14 Aug 2015
23. Protégé ontology editor: http://protege.stanford.edu/. Accessed 14 Aug 2015
24. SPARQL 1.1 overview: http://www.w3.org/TR/sparql11-overview/. Accessed 14 Aug 2015