Applying Semantic Web Technologies to Context Modeling in Ambient Intelligence

Alexandru Sorici[1,2], Olivier Boissier[1], Gauthier Picard[1], and Antoine Zimmermann[1]

[1] Ecole Nationale Supérieure des Mines, FAYOL-EMSE, LSTI , F-42023 Saint-Etienne
{sorici,boissier,picard,antoine.zimmermann}@emse.fr
[2] University Politehnica of Bucharest, Department of Computer Science, 313 Splaiul Independentei, 060042 Bucharest, Romania

Abstract. Representation and reasoning about context information is a main area of research in Ambient Intelligence (AmI). Given the openness and decentralization of many AmI applications, we argue that usage of semantic web technologies for context modeling brings advantages in terms of standards, uniform representation and expressive reasoning. We present an approach for modeling of context information which builds and improves upon related lines of work (SOUPA, CML, annotated RDF). We provide a formalization of the model and an innovative realization using the latest proposals for semantic web standards like RDF and SPARQL. A commonly encountered ambient intelligence scenario showcases the approach.

Keywords: Ambient Intelligence, Context Modeling, Ontologies, Semantic Web

1 Introduction

Ambient Intelligence is nowadays a well recognized area of research with work done in domains ranging from hardware (e.g. sensors, actuators) through middleware (e.g. information management, basic services) to innovative end applications and human computer interfaces. Ambient Intelligence applications are very often open and decentralized, involve a large number of heterogeneous devices and have to handle large amounts of information. Thus, the information management middleware and the notions of context representation and reasoning, as introduced by Dey [1], are very important. The past decade has seen many contributions in this particular field of research [3, 5]. Due to the open and heterogeneous nature of ambient intelligence scenarios, recent works have recognized the need for interoperability and standards in terms of languages and approaches, thereby focusing on ontology models in support of context management. Amongst the proposed ontologies ever more [2, 4] are seeing the need to offer explicit modeling of meta-properties of context information like accuracy or temporal validity. Few models however end up showing concrete ways of representation and reasoning for both context information and its meta-properties.
We therefore propose an extensive ontology model for representation of context, which extends and combines previous work [8, 12] and handles the earlier presented aspects. Furthermore, we argue that the decentralized systems of ambient intelligence would benefit greatly from advances in semantic web technologies because they can help with standardization and scalability. We therefore propose a new way of storing, querying and reasoning over context information and its meta-properties which uses the latest proposals of the semantic web community for standards like

RDF and SPARQL.
In Section 2 we introduce supporting concepts and review previous work that inspired our own approach. We continue in Section 3 with a formal specification of our context representation and reasoning model and show how it maps to an ontology definition and use of semantic web technologies in Section 4. We provide an example scenario which highlights our approach in Section 5 before concluding in Section 6.

2 Background

This section reviews the research work which serves as background for our own proposal and shows how we build and extend upon it.
The field of context modeling has received a noticeable amount of contributions in the last decade [3, 5] ranging from simple key-value models, through markup and graphical models [10, 11] down to different ontology models [2, 4, 8]. As mentioned earlier, ontologies for context modeling are becoming a focus in many approaches and the authors in [9] note that the Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) [8] is one of the proposals that is often reused in other projects. Indeed, SOUPA is built in a modular way and reuses in its core many well established ontologies for modeling entities like persons (FOAF[3]), places (spatial ontologies in OpenCyc[4]), security or privacy policies (Rei policy ontology[5]) or even an agent's beliefs desires or intents (BDI). Though SOUPA was originally used in applications centered on user activity modeling, it is general enough such that it can be used as an upper-ontology for describing the entities involved in a wider variety of context management applications. Still, SOUPA does not provide a way to further characterize the type of properties defined between its core classes, nor does it offer direct support for annotations of these properties.
One approach which offers some answers to the above mentioned drawbacks is the Context Modeling Language (CML) model proposed by Henricksen [11]. CML builds upon the Object-Role Model (ORM) conceptual language for data modeling used in the Relational Database domain and extends it with constructs specific to the area of context representation. An important aspect of CML is that its basic representational unit is the *fact*, a relationship that holds between one or more entities. The model then provides a framework for classification of facts into categories denoting its type (static, sensed, profiled, derived, temporal), expression of uniqueness constraints and fact dependencies as well as annotation of facts with quality indicators (e.g. freshness, accuracy). CML also introduces a form of first-order predicate logic used to derive higher-level information called *situation* of entities based on the currently existing facts. One drawback of the model though is that it does not provide a base for the entities involved in the facts and its representation fails to easily capture potential subclassing hierarchies of entities or even facts themselves. Additionally, CML focused on custom relational database techniques for model realization and it needed to provide its own implementation for the situation definition language. This could lead to difficuly in standardization of representation and scaleability. In contrast, our approach argues that semantic web technologies can mitigate these drawbacks. We propose a combination between an adapted version of SOUPA as the upper-ontology for modeling of application entities and an ontology model and RDF triple based realization that incorporates the strengths of CML. We also introduce an expressive derivation language that can be translated into equivalent

[3] http://www.foaf-project.org
[4] http://www.cyc.com/opencyc
[5] http://rei.umbc.edu/

SPARQL queries. We detail this proposal in Section 4.
Lastly, we noted earlier that an increasing number of works recognize the importance of modeling meta-properties like different measures of quality or provenance of context information. Yet, few of them focus on a systematic way of representing and reasoning about these annotations. On the other hand, work in the field of annotated RDF statements [6, 7] provides formalization in terms of algebraic structures for the annotation language and the corresponding deductive system. Zimmermann et al. [6], for instance, define the semantics of a set of generic operators which are used to combine the annotations of statements during RDFS inferencing. They provide instantiations of these operators and the accompanying algebraic structures for the time, fuzzyness and provenance annotation domains. These works however do not concentrate on a concrete realization of their abstract annotation structures. As we detail further in Sections 3.3 and 4.3 our approach to context modeling adopts the more rigorous definition of annotation dommains proposed by work in annotated RDF while also explaining the mean for concrete implementation of the discussed meta-properties.
In summary, the context representation and reasoning model we detail in the following sections tries to combine the best of the 3 approaches presented above.

3 Context Representation

In this section we present a formalization of our context representation and reasoning models and then show (in section 4) their implementation using semantic web technologies.

3.1 Context Model Concepts

Let us first introduce the key concepts participating in the definition of the context model (cf. Figure 1). A *ContextAssertion* represents an informational fact, a basic truth unit used to describe the context situation of entities (e.g. a "person", a "place", an "object") "which are considered relevant for the interaction between a user and an application" (in the sense of Dey's definition of context [1]). Each assertion is therefore a relation involving one or more *ContextEntities*. As a short example consider the following relation of the form: *nearDevice(bob, lcdScreen)*. The *ContextAssertion* "nearDevice" involves two *ContextEntities*: *Person*("bob") and *Device* ("lcdScreen").

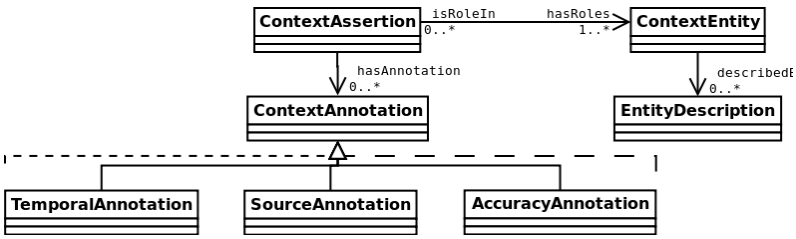

Fig. 1: The defining concepts of our proposed context model.

ContextAssertion (resp. *ContextEntity*) may be characterized by additional meta-properties through *ContextAnnotation* (resp. *EntityDescription*). Annotations of assertions can be their source (author of the statement), the timestamp of their generation, their validity (time intervals for which the assertion is considered to be true) or their accuracy. Since our model doesn't limit their number, other annotations can be imagined such as ownership (one or more entities which "hold control" of an assertion), access control (who is allowed to query or access the value of the assertion) or other.
EntityDescriptions are binary relations amongst *ContextEntities* and/or literals which don't need to be further annotated (as opposed to *ContextAssertions*). For instance, a context engineer can model $hasContactProfile(bob, bob\_cp)$ as a *ContextAssertion* where "bob_cp" is an instance of *ContactProfile* (a subtype of *ContextEntity*). Relations like $email(bob\_cp, emailBob)$ or $homepage(bob\_cp, homepageBob)$ are then modeled as additional descriptions of a *ContactProfile* instance (where in this case "emailBob" and "homepageBob" are literals).

3.2 ContextAssertion

Let us now give the formal definition of *ContextAssertion*. In the following, E denotes the set of *ContextEntities*, V the alphabet of variables, L the alphabet of literals and A_d the one of a *ContextAnnotation* domain d (which we discuss further in section 3.3). Further, let $\mathcal{F}$ be the set of all *ContextAssertions* and $\mathcal{A}$ be the set of all *ContextAnnotation* domains A_d.
A *ContextAssertion* F is then defined as an n-ary relation of the form $F(x_1, x_2, \ldots, x_n) : \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$, where $x_i \in E \cup L \cup V, i = 1..n$ and $\lambda_j \in A_{d_j}, j = 1..m$.
The function $role : \mathcal{F} \rightarrow 2^{E \cup L \cup V}$ returns the set of *ContextEntities*, literals or variables which play a *role* in an assertion, i.e. $role(F) = \{x_1, \ldots, x_n\}$.
Similarly, the function $ann : \mathcal{F} \rightarrow 2^{\mathcal{A} \cup V}$ returns all *ContextAnnotations* (instances or variables) which describe a *ContextAssertion* F, i.e. $ann(F) = \{\lambda_1, \ldots, \lambda_m\}$.
To each *ContextAssertion* we may also attach *key* and *uniqueness constraints* which are similar to the ones defined for the Relational Database domain and serve the same purpose of keeping the knowledge base consistent. It is important to note that we allow these constraints to be defined over a subset $K \subseteq role(F) \cup ann(F)$ of both context entities and annotations, meaning that annotations are important in determining the consistency of *ContextAssertions*.

3.3 ContextAnnotations

The definition of a *ContextAnnotation domain* A_d is inspired from work on annotated RDFS [6] and is defined to have the structure of an idempotent, commutative semi-ring $\langle A_d, \oplus, \otimes, \bot, \top \rangle$. Within this algebraic structure, the operators $\oplus$ and $\otimes$ can be considered similar to the meet and join operators of a bounded lattice. Following the observation in [6], we use $\oplus$ to combine information about the same statement, whereas $\otimes$ models the conjunction of two annotated statements. In what follows, we shortly present the form of the annotation domains for the common settings we discussed earlier in this section. We provide a more detailed working example of both annotated *ContextAssertions* and reasoning with those annotations in Section 5. The usually encountered annotation settings have the following domain definitions (partly adopted from [6]):

- source: $A_{source} =$ set of URIs $\cup \{none, all\}$, $\langle A_{source}, \vee, \wedge, none, all \rangle$

- timestamp: $A_{timestamp}$ = set of datetime strings down to ms precision, $\langle A_{timestamp}, max, min, -\infty, +\infty \rangle$
- time validity: $A_{validity}$ = the set of sets of pairwise disjoint time intervals, $\langle A_{validity}, \cup, \cap, \{\emptyset\}, \{[-\infty, +\infty]\} \rangle$
- accuracy: $A_{accuracy} = [0, 1]$, $\langle A_{accuracy}, max, min, 0, 1 \rangle$

3.4 ContextAssertion Derivation Rules

We now focus on the proposed reasoning model. It aims at combining several *EntityDescriptions*, *ContextAssertions* and their annotations in order to obtain higher-level *ContextAssertions*. The reasoning is based on a deduction mechanism involving *Context Derivation Rules*. Each rule is made up of a head (the deduced *ContextAssertion*) and a body which expresses the conditions required for the rule head to be deduced.
The head of a derivation rule ρ is a *ContextAssertion* $F_{head}(x_1, \ldots, x_k) : \{\lambda_1, \ldots, \lambda_l\}$ where $x_i \in E \cup V \cup L$ and $\lambda_j \in A_{d_j} \cup V$. Notice that $role(F_{head})$ and $ann(F_{head})$ can include variables which will be bound during the reasoning process.
The body consists of so called *ConditionExpressions* (detailed further down in this section) and constrained forms of universal and existential quantification. The general form of a *Context Derivation Rule* is thus the following:

$$\begin{array}{lll} \rho : F_{head}(x_1, x_2, \ldots, x_k) : \{\lambda_1 \lambda_2, \ldots, \lambda_l\} \leftarrow body & & \text{where body may be:} \\ & ConditionExpr & \\ or & \exists y_1, \ldots y_r \bullet F_c(z_1, \ldots, z_m) : \{\lambda_1, \ldots \lambda_p\} \bullet ConditionExpr & (EQC) \\ or & \forall y_1, \ldots y_r \bullet F_c(z_1, \ldots, z_m) : \{\lambda_1, \ldots \lambda_p\} \bullet ConditionExpr & (UQC) \end{array}$$

where $y_i \in V$, $Y_\rho = \{y_1, \ldots y_r\} \subseteq role(F_c) \cup ann(F_c)$ and $role(F_{head}) \cap role(F_c) \neq \emptyset$. In the above rule, EQC (resp. UCQ) refers to existential (resp. universal) quantification constraint, meaning that the possible values of the variables y_i are those for which the constraining *ContextAssertion* F_c is true. In the existential case, at least one value assignment for each y_i has to also respect the conditions set in *ConditionExpr*. In the universal case, all possible value assignments have to do so. Additionally, Y_ρ and all the variables that appear in the rule head ($role(F_{head})$, $ann(F_{head})$) must also appear in *ConditionExpr*, which we discuss next.

A *ConditionExpression* contains a domain expression (*DomExpr*) and an annotation expression (*AnnExpr*) as follows:

$$\begin{aligned} ConditionExpr &::= DomExpr \wedge AnnExpr \\ DomExpr &::= ComExpr \mid DomExpr \wedge ComExpr \\ ComExpr &::= SimExpr \mid AggExpr \\ SimExpr &::= AssertionExpr \mid \neg AssertionExpr \mid TermExpr \\ AggExpr &::= aggregate(FuncExpr, FilterExpr, ResExpr) \\ AssertionExpr &::= F_{body}(x_1, \ldots, x_n) : \{\lambda_1, \ldots, \lambda_m\},\ x_i \in E \cup V \cup L,\ \lambda_j \in A_{d_j} \cup V \end{aligned}$$

A *DomExpr* is a conjunction of positive or negated *ContextAssertions*, term expressions (*TermExpr*) and aggregations (*AggExpr*).
Term expressions contain terms $t \in E \cup L \cup V \cup A_{d_i}$ which are entities, literals, variables or annotations. Terms can be related by boolean operators ($>$, $<$, $\geqslant$, $\leqslant$, $=$, $\neq$), logical connectors ($\wedge$, $\vee$, $\neg$), *EntityDescription* relations and system or user-defined functions $func(t_1, \ldots, t_n)$. Functions in term expressions act like predicates

which return a truth value when all their arguments are bound. If the arguments contain free variables, the function call binds them to values that make the function true.
An *AggExpr* contains three subexpressions: $FuncExpr$, $FilterExpr$ and $ResExpr$. The $FuncExpr$ is a list of one or more aggregation functions that take a single variable as their argument $[aggFunc_1(z_1), \ldots, aggFunc_k(z_k)]$. We employ the typical aggregation functions: $aggFunc \in \{count, sum, avg, min, max\}$. $FilterExpr$ is the expression used to condition the values of the variables z_i over which we perform the aggregation. Its form is the same as that of the $SimExpr$. Therefore, all variables z_i must occur in *ContextAssertions* or *ContextAnnotations* contained in $FilterExpr$. Finally, $ResExpr$ is a list of variables $[aggRes_1, \ldots, aggRes_k]$ which will store the result of the k $aggFunc_i(z_i)$ functions.
An *AnnExpr* is a conjunction of functions of the form $f_j(\lambda_{j1}, \ldots, \lambda_{jq})$ where each function f_j binds a free variable $\lambda_j^{head} \in ann(F_{head})$ (the annotations of the *ContextAssertion* in the rule head). All λ_{jk} and λ_j^{head} belong to the same annotation domain A_{d_j} and we additionally know that λ_{jk} belongs to the annotations of some *ContextAssertion* in $ConditionExpr$. The functions f_j are user-defined and can bind λ_j^{head} either to a constant (e.g. the default value for annotation domain A_{d_j}) or to a value computed using the $\oplus$ and $\otimes$ operators specific to each annotation domain.

4 Implementing the model with Semantic Web Technologies

The formal model introduced in the previous section has been implemented using available semantic web technologies. In what follows we first present the ontology used to express the *ContextEntities* and the *ContextAssertions*. We then show how assertions and their annotations are inserted in an RDF quad store using the SPARQL query language[6] and the concept of Named Graphs. We then finish by detailing how and the way in which uniqueness constraints and context derivation rules are implemented using the SPARQL inferencing notation (SPIN).

4.1 Mapping ContextEntities and ContextAssertions to Ontology Models

In section 2 we discussed about SOUPA [8] as one of the most renowned and reused ontologies for representing context information [9]. Its modularity and reuse of existing vocabularies provides a good incentive to adopt it as the upper-ontology of our context modeling approach. We extend the original definition of SOUPA with a number of classes and properties which allow us to better express the particularities of our representation and reasoning models. These new concepts are part of an ontology we call ContextAssertion.
In the ContextAssertion ontology we define the generic class ContextEntity and use it as the new root for all classes existing in SOUPA. In section 3.1 we introduced the concepts of *ContextAssertion* and *EntityDescription* which are relations over *ContextEntities* and literals. In order to characterize the binary assertions and descriptions which already exist between classes of SOUPA, in the ContextAssertion ontology we define two OWL object properties (entityRelationAssertion, entityRelationDescription) and two datatype properties (entityDataAssertion, entityDataDescription) that help us "classify" SOUPA's object

[6] http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

and datatype properties as either *ContextAssertion* with arity $n = 2$ (subproperties of ∗Assertion) or *EntityDescription* (subproperties of ∗Description).
Taking inspiration from work in [11], we further extend entityRelationAssertion and entityDataAssertion into static, sensed, profiled or derived properties, denoting the way in which those assertions are obtained within the system.
To express *ContextAssertions* with arities $n = 1$ or $n \geqslant 3$ we introduce two new classes of the ContextAssertion ontology: UnaryContextAssertion ($n = 1$) and NaryContextAssertion ($n \geqslant 3$).
For the unary case, a *ContextAssertion* like *occupied*(*bob*) entails the creation of the Occupied subclass of UnaryContextAssertion and the assertion of the statement `:bob a :Occupied` (in Turtle syntax). In the $n \geqslant 3$ case we make use of a mechanism which is similar to reification of RDF statements[7]. In the ContextAssertion ontology we define the assertionRole property relating an instance of a NaryContextAssertion to a ContextEntity or literal which plays a role in the assertion. In order to express a *ContextAssertion* like *personInMeetingAt(bob, teamMeeting, room123)*, we first create the PersonInMeetingAt subclass of NaryContextAssertion together with subproperties of assertionRole specifying its roles (personRole, meetingRole, locationRole). The assertion in our example is then expressed as the following group of statements: {`:_a0 a :PersonInMeetingAt. :_a0 :personRole :bob. :_a0 :meetingRole :teamMeeting. :_a0 :locationRole :room123.`}, where `_a0` is an RDF blank node.
For instances of UnaryContextAssertion and NaryContextAssertion we also define an assertionType property which states if they are static, sensed, profiled or derived (like in the $n = 2$ case).

4.2 Implementing ContextAssertions

Let us now discuss how we complete *ContextAssertions* expressed using the ContextAssertion and adapted SOUPA ontologies with the ability to become annotated statements. In doing so we make use of the concepts of quad stores and RDF named graphs[8,9]. RDF named graphs are a key concept of the Semantic Web. They allow a set of RDF statements (subject - predicate - object triples) called a graph to be identified by a URI. By naming the set of statements, additional meta-properties can then be asserted about the entire set.
We use this facility to our advantage with the purpose of expressing annotations of a *ContextAssertion*. We allow each individual RDF statement ($n = 1, 2$) or set of statements ($n \geqslant 3$) expressing a *ContextAssertion* to be wrapped within its own graph. Essentially, the graph URI becomes an identifier for the *ContextAssertion*. The *ContextAnnotations* are then expressed as RDF statements which have the graph URI (the identifier) as the subject.
Considering now an entire context model expressed in terms of the adapted SOUPA and ContextAssertion ontologies, our system performs the following steps for setting up and maintaining the named graph structure of the context assertions and their annotations. First, the RDF file containing the *ContextEntity* and *ContextAssertion* definitions is obtained and parsed. A SPARQL `CREATE GRAPH <baseURL/#ContextEntityStore>` statement is issued to create a named graph which will store all instances of ContextEntity subclasses as well as all *EntityDescriptions* expressed with subproperties of entityRelationDescription or entityData-

[7] http://www.w3.org/TR/rdf-mt/#Reif
[8] http://en.wikipedia.org/wiki/Named_graph
[9] http://www.w3.org/TR/sparql11-query/#specifyingDataset

Description. Next, for each type of *ContextAssertion* we use similar SPARQL statements to create a named graph with an identifier URI that follows a pattern (e.g. for an assertion called *nearDevice* we build the `#nearDeviceStore` relative URI). These named graphs will hold all *ContextAnnotations* made for instances of their respective type of assertions. Lastly, after building this supporting structure of named graphs, insertion of a new *ContextAssertion* requires the following actions: *(i)* issue a `CREATE GRAPH` request with a unique URI naming the graph that will wrap the new assertion and act as its identifier; *(ii)* use a `SPARQL INSERT` request to put the necessary RDF triples stating the assertion in the newly created named graph. *(iii)* issue `SPARQL INSERT` requests for *ContextAnnotations* in the named graph corresponding to the name of our assertion as seen above. The graph URI created at step (i) serves as the subject of the RDF triples expressing the annotations.

4.3 Implementing ContextAnnotations

We now describe our concrete representation for the annotation domains discussed throughout this article (source, timestamp, time validity and accuracy).
As described in Section 3.3, the vocabulary for the source annotation domain consists of the set of URIs identifying authors of *ContextAssertions*. In the ContextAssertion ontology we define the ContextAgent class, which describes the different type of actors (agents) that we envision within the domain of discourse of an application (e.g. ContextSensingAgent, ContextUserAgent, etc). Thus, A_{source} is equal to the URIs identifying instances of the ContextAgent class.
For the timestamp annotation domain, the vocabulary $A_{timestamp}$ consists of the set of datetime strings. The RDF schema specification defines a rdfs:Datatype of `xsd:datetime` which suits our needs accordingly.
In the case of the time validity annotation domain, an element of the vocabulary $A_{validity}$ is a set of pairwise disjoint time intervals. We employ the "entry" sub-ontology of time in OWL[10] [13] to express a time interval using the ProperInterval class. In order to express a set of intervals we make use of the Ordered List Ontology[11] which describes a list in terms of Slot classes and next and previous properties. A time validity annotation is then an instance of the OrderedList class where each validity interval fills a slot.
Lastly, for the accuracy annotation domain the vocabulary consists of float values in the continuous interval $[0, 1]$. We can express them using the `xsd:float` datatype.
A named graph URI identifying a *ContextAssertion* is related to a particular *ContextAnnotation* instance by the assertedBy, hasTimestamp, validDuring and hasAccuracy properties defined in the ContextAssertion ontology for the source, timestamp, time validity and accuracy annotation domains respectively.
For each annotation domain, the respective $\oplus$ and $\otimes$ operators either already exist in the operational semantics of SPARQL ($\vee$, $\wedge$, max, min) or they are implemented as user-defined functions ($\cup$ and $\cap$ for the time validity domain).

4.4 Implementing ContextAssertion Uniqueness Constraints

Uniqueness constraints imposed on a *ContextAssertion* are enforced using SPIN constraints[12]. To explain, consider the example of a *personRoomLocation ContextAssertion*. A SPARQL ASK query saying that a person cannot be in two places of a room at the same time looks like the following:

[10] http://www.isi.edu/ hobbs/damltime/time-entry.owl
[11] http://smiy.sourceforge.net/olo/spec/orderedlistontology.html#
[12] http://spinrdf.org/spin.html#spin-constraints

```
ASK WHERE {
  GRAPH ?g1 {?P :personRoomLocation ?RoomLoc1} .
  GRAPH <:personRoomLocationStore> { ?g1 :validity ?v1 }.
  FILTER (
    EXISTS {
      GRAPH ?g2 {?P :personRoomLocation ?RoomLoc2}.
      GRAPH <:personRoomLocationStore> { ?g2 :validity ?v2 }.
      FILTER ( ?RoomLoc1 != ?RoomLoc2 && fn:overlaps(?v1, ?v2)).
    }
  )
}
```

The above query will return true when there is an overlap between the stated time validities of personRoomLocation assertions relating the same person *?P* to different locations inside the room (*?RoomLoc1*, *?RoomLoc2*). The SPIN specification allows such queries to be attached to an OWL class definition using the spin:constraint property. In our system, for a given *ContextAssertion* (e.g. personRoomLocation) we attach such a constraint to all ContextEntity classes which play a role in the assertion and are involved in the uniqueness constraint (e.g. the Person class in our example).

4.5 Implementing ContextAssertion Derivation Rules

To map the derivation rules which drive the reasoning process we again use SPIN which allows attaching different types of SPARQL queries to a subclass of rdfs:Class. Recall from Section 3.4 that the head of a derivation rule ρ is a *ContextAssertion* $F_{head}(x_1, \ldots, x_k) : \{\lambda_1, \ldots, \lambda_l\}$. We use the spin:rule[13] property to attach the corresponding body of ρ, expressed in SPARQL syntax, to a *ContextEntity* instance $x_i \in role(F_{head})$. At runtime, a SPIN compliant inference engine first searches all *ContextEntities* for attached *Derivation Rules*. It then inspects the rules to build a dictionary mapping all *ContextAssertions* to rule bodies where they appear. The engine can then execute those rule bodies every time the value of a mapped *ContextAssertion* changes. We now show how the *ConditionExpr* is expressed in terms of SPARQL requests:

- *AssertionExpr*: a *ContextAssertion* is expressed using RDF statements wrapped in named graphs as explained in Sections 4.1 and 4.2
- *AggExpr*: are expressed using SPARQL aggregates[14]
- *TermExpr*: *EntityDescriptions* are expressed as RDF statements encoding the binary description relations. Boolean operations, logical connectives and functions on terms are implemented using the equivalent SPARQL syntax and contained within SPARQL FILTER expressions.
- *AnnExpr*: the annotation assignment functions are user-defined. The value they compute is bound to the corresponding λ_j variable in the rule head ($ann(F_{head})$) using a SPARQL BIND statement.

Finally, let us consider the existentially and universally constrained quantifications. For the existential case support is already provided in SPARQL by the EXISTS filter expression (see Fig. 2). For the universal case the intuition behind the SPARQL query shown in Fig. 2 is the following: consider a substitution $\sigma = \{y_1/t_1, \ldots, y_r/t_r\}$ which binds each variable $y_i \in Y_\rho$ to a *ContextEntity* or literal. Let us then denote by $\Sigma_{F_c \downarrow Y_\rho}$ and $\Sigma_{DerivExpr \downarrow Y_\rho}$ the sets of all substitutions σ binding variables in Y_ρ for which the constraining assertion F_c and the assertions in *ConditionExpr* are true

[13] http://spinrdf.org/spin.html#spin-rules
[14] http://www.w3.org/TR/sparql11- query/#aggregates

respectively. The interpretation of the universal constrained quantification rule then implies that $\Sigma_{F_c \downarrow Y_\rho} \subseteq \Sigma_{DerivExpr \downarrow Y_\rho} \Leftrightarrow \Sigma_{F_c \downarrow Y_\rho} \setminus \Sigma_{DerivExpr \downarrow Y_\rho} = \emptyset$. The SPARQL MINUS[15] filter expression used in Fig. 2 provides this exact semantics.

```
CREATE GRAPH <gURI>;
INSERT{
  GRAPH <gURI> {new assertion}
  GRAPH <newAssertionStore> {annotations}
}
WHERE {
 { constraining assertion } .
 FILTER (
  EXISTS { ConditionExpr }
 )
}
```

```
CREATE GRAPH <gURI>;
INSERT{ assertion and its annotations }
WHERE {
 { SELECT (COUNT(*) AS ?count)
    WHERE {
     {constraining assertion}
     MINUS
     {ConditionExpr}
    }
 } . FILTER (?count = 0)
}
```

Fig. 2: SPARQL expressions for existentially (left) and universally (right) constrained quantifications

5 A sample Use Case

To exemplify both our formal context model and its presented implementation with semantic web technologies we employ a fairly known ambient intelligence scenario: the ad-hoc meeting in a smart office. Alice, Bob and Charlie enter the smart office

```
ContextEntity:{camera, microphone,
 roomLocation}
Literal:{noise level (in dB),
 skeleton position}
ContextAssertion: {
 deviceRoomLoc(D, RL),
 sensesSkelInPos(Cam, S, P):{λ_valid, λ_acc},
 hasNoiseLevel(RL, NL):{λ_valid, λ_acc},
 hostsAdhocMeeting(RL):{λ_valid, λ_acc}
}
```

```
:Device rdfs:subClassOf :ContextEntity
:RoomLoc rdfs:subClassOf :ContextEntity
:Kinect rdfs:subClassOf :Device
:Mic rdfs:subClassOf :Device
:deviceRoomLoc rdfs:subPropertyOf :entityRelationAssertion
:SensesSkelInPos rdfs:subClassOf :NaryContextAssertion
:hasNoiseLevel rdfs:subPropertyOf :entityDataAssertion
:HostsAdhocMeeting rdfs:subClassOf :UnaryContextAssertion
```

Fig. 3: Formalization of the ad-hoc meeting scenario

in room 123 to have a meeting and discuss an important issue. The office walls are lined with Microsoft Kinect cameras and microphones near every desk. Sensed data from these devices is continuously processed by a computing agent which monitores the situations in the room. As the 3 friends sit down and start discussing, the Kinect camera near their desk observes 3 skeletons in the *sitting* position and the microphones in the vicinity inform of a signal level which is constantly over 60dB. Since this is going on for more than 5 minutes, the computing agent of room 123 recognizes it to be an *ad-hoc meeting situation* and relays this information to the user agents running on the smartphones of Alice, Bob and Charlie. Using customized rules these agents can decide to silence the ringtone or divert all calls to voice-mail as long as the meeting is in progression.
We focus on the *ContextAssertions* required to model this scenario and on the reasoning required to make the assertion that a desk in the smart office is host to an *ad-hoc meeting*. Figure 3 shows the formalization and representation within the SOUPA and ContextAssertion ontologies. In order to capture the described target situation we need to model the *ContextEntities* (*Camera, Microphone, RoomLocation*)

[15] http://www.w3.org/TR/sparql11-query/#neg-minus

```
hostsAdhocMeeting(RL):{λ_src, λ_t, λ_valid, λ_acc}:

  ∃K• deviceRoomLoc(K, RL) •
  isA(K, camera) ∧
  makeInterval(now()-5,now(),λ_interv) ∧
  aggregate([count(S),avg(λ_accS)],
    sensesSkelInPos(K,S,sit):{λ_validS, λ_accS}
    ∧includes(λ_validS, λ_interv),[Ct,avgAccS]
  ) ∧ Ct⩾3 ∧ λ_avgAccS ⩾0.75 ∧
  ...
  ...
  aggregate([avg(NL),avg(λ_accN)],
    hasNoiseLevel(RL, NL):{λ_time, λ_accN} ∧
    λ_time ⩾ now()-5 ∧ λ_time < now(),
    [AvgLevel,λ_AvgAccN]
  ) ∧ AvgLevel⩾60 ∧ λ_AvgAccN ⩾ 0.75 ∧
  assignAcc(λ_acc, λ_AvgAccS ⊗ λ_AvgAccN) ∧
  assignSrc(λ_src, currentAgent) ∧
  assignTimestamp(λ_t, now()) ∧
  assignValid(λ_valid, {[now()-5, now()]})
```

Fig. 4: Ad-hoc Meeting Context Derivation Rule

and the *ContextAssertions* that relate them (*deviceRoomLocation, hasNoiseLevel, sensesSkelInPosition*). Notice that all 3 types of assertions ($n = 1, 2$ and $n \geqslant 3$) are present. Figures 4 and 5 present the *Context Derivation Rule* used to detect the ad-hoc meeting situation. We can observe again that all elements described throughout Sections 3 and 4 are present: *assertion expressions, aggregation expressions, term expressions* including predefined system functions (e.g. $fn : now()$ or $fn : includes()$ which determines interval inclusion), as well as examples of functions in *AnnExpr* like $fn : multiply()$ (implementing the semantics of *assignAcc* in Figure 4) which sets the value λ_{acc} for the assertion in the rule head. At the beginning of the SPARQL query we can also see the sequence of named graph creation and statement insertions discussed about in Section 4.2.

```
create graph <UUID>;
insert {
  graph <UUID> {?RL a :HostAdhocMeeting}
  graph <HostAdHocMeetingStore> {
   <UUID> :assertionType :Derived;
          :assertedBy ?src;
          :hasTimestamp ?t;
          :validDuring ?valid;
          :hasAccuracy ?acc.    }
} where {
  ?K :deviceRoomLoc ?RL .
  bind (fn:makeInterval(fn:now()-5,
   fn:now())) as ?interv.
  FILTER (
   EXISTS { ?K a ctx:KinectCamera .
    {select (count(?S) as ?Ct)
      (avg(?accS) as ?AvgAccS)
      where {
        graph ?gCamera {
          _n a :SensesSkelInPos;
               :cameraRole ?K;
               :skelRole ?S;
               :posRole "sit"
        }.
  .......
        graph <SensesSkelInPosStore> {
          ?gCamera :validDuring ?validS;
                   :hasAccuracy ?accS
        }.
        filter(fn:includes(?validS, ?interv))
      }.
    }.
    filter(?Ct>=3 && ?AvgAccS>=0.75).
    {select (avg(?lvl) as ?AvgLvl)
      (avg(?accN) as ?AvgAccN)
      where {
        graph ?gNoise {?RL :hasNoiseLevel ?lvl}.
        graph <hasNoiseLevelStore> {
          ?gNoise :hasTimestamp ?time;
                  :hasAccuracy ?accN
        }.
        filter (?time>=fn:now()-5 && ?time<fn:now()).
      }
    }. filter(?AvgLvl>=60 && ?AvgAccN>=0.75).
   }
  ). bind (fn:multiply(?AvgAccS, ?AvgAccN) as ?acc).
     bind (fn:currentAgent() as ?src).
     bind (fn:now() as ?t).
     bind (?interv as ?valid)
}
```

Fig. 5: SPARQL query mapping of Ad-hoc Meeting Context Derivation Rule

6 Conclusions and Future Work

In this article we presented a model for representation and reasoning over context information in ambient intelligence scenarios. We provided a formalization of the concepts involved in the model and have shown how they can be concretely implemented using an extended ontology (SOUPA + ContextAssertion) and the newest in semantic web technologies (usage of RDF Named Graphs and SPIN). Our proposed rule based reasoning model features a rich assertion derivation language

capable of handling constrained quantification, aggregation and manipulation of context information annotations. It can therefore handle reasoning over complex situations involving things like averages over sequences of events and decisions over uncertain information. Additionally, the usage of ontologies and technologies of the semantic web offers our model advantages like interoperability, incremental knowledge build-up and decentralized control of information.
In future work we will test our model over different RDF quad stores and benchmark its performance in terms of resource consumption, reasoning throughput and delay. We plan to extend *ContextAnnotations* with domains we hinted towards (e.g. ownership, access control) and we will develop a query and subscription language which takes different policies (e.g. source of information, update frequency, access control) into account.

References

1. Dey, Anind K.: Understanding and using context. Personal and ubiquitous computing 5.1, pp. 4–7 (2001).
2. Strang, T., Linnhoff-Popien, C., Frank, K.: CoOL: A context ontology language to enable contextual interoperability. Distributed applications and interoperable systems. 236–247 (2003).
3. Strang, T., Linnhoff-Popien, C.: A context modeling survey. In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004-The Sixth International Conference on Ubiquitous Computing, Nottingham/England. (2004).
4. Gu, T., Wang, X., Pung, H.: An ontology-based context model in intelligent environments. In: Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference (2004).
5. Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D.: A survey of context modelling and reasoning techniques. Pervasive and Mobile Computing. 6, 161–180 (2010).
6. Zimmermann, A., Lopes, N., Polleres, A., Straccia, U.: A general framework for representing, reasoning and querying with annotated semantic web data. Web Semantics: Science, Services and Agents on the World Wide Web, 11, 72–95 (2012)
7. Geerts, F., Karvounarakis, G., Christophides, V., Fundulaki, I.: Algebraic structures for capturing the provenance of SPARQL queries. In: Proceedings of the 16th International Conference on Database Theory. pp. 153–164 (2013).
8. Chen, H., Perich, F., Finin, T.: SOUPA: Standard ontology for ubiquitous and pervasive applications. In: Mobile and Ubiquitous Systems: Networking and Services. (2004).
9. Krummenacher, R., Strang, T.: Ontology-based context modeling. In: Proceedings of Third Workshop on Context-Aware Proactive Systems (CAPS). (2007).
10. Indulska, J., Robinson, R., Rakotonirainy, A., Henricksen, K.: Experiences in using cc/pp in context-aware systems. In: Mobile Data Management. pp. 247–261 (2003).
11. Henricksen, K.: A Framework for Context-Aware Pervasive Computing Applications. PhD Thesis, School of Information Technology and Electrical Engineering, University of Queensland. (2003).
12. Henricksen, K., Livingstone, S., Indulska, J.: Towards a hybrid approach to context modelling, reasoning and interoperation. In: Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning And Management, in conjunction with UbiComp (2004).
13. Pan, F., Hobbs, J. R.: Time in OWL-S. In: Proceedings of AAAI-04 Spring Symposium on Semantic Web Services. (2004).