Journal of Ambient Intelligence and Smart Environments 1 (2009) 1–5
IOS Press

Conflict resolution when axioms are materialized in semantic-based smart environments.

Christophe Gravier [a,*], Julien Subercaze [a] and Antoine Zimmermann [b]
[a] *Université de Saint-Étienne, Télécom Saint-Étienne, 23 rue du Dr. Rémy Annino, F-42023 Saint-Etienne, France*
E-mail: {christophe.gravier, julien.subercaze}@univ-st-etienne.fr
[b] *École Nationale Supérieure des Mines, FAYOL-ENSMSE, LSTI, F-42023 Saint-Étienne, France*
E-mail: {antoine.zimmermann}@emse.fr

Abstract. In Semantic Web applications, reasoning engines that are data intensive commonly materialise inferences to speed up processing at query time. However, in evolving systems, such as smart environments, semantic-based context aware systems (SCAS) [6] or social software with user-generated data, knowledge does not grow monotonically: newer facts may contradict older ones, knowledge may be deprecated, discarded or updated such that knowledge must sometimes be retracted. We are describing a technique to retract explicit and inferred statements, when some information becomes obsolete, as well as retracting any statement that would lead to get back the removed explicit statements. This technique is based on OWL justifications and is triggered whenever a knowledge base becomes inconsistent, such that the system stays in a consistent state all the time, in spite of uncontrolled evolution. We prove termination and correctness of the algorithm, and describe the implementation and evaluation of the proposal.

Keywords: Ontologies, Reasoning, Explanations, Agents, Ambient Intelligence, Smart Environments

Introduction

For a long time, semantic web technologies were only providing standards and tools for static, monolithic knowledge bases (hereafter abbreviated KB). Only recently the ability to update RDF stores [1], to process RDF streams have been tackled, but a final standard only appeared very recently [2] In a situation where appending, modifying or deleting can be performed, reasoning has to be carried on in a fairly different manner. RDF databases, also called *triple stores*, with high capacities are often able to perform reasoning at large scale with good response time to queries due to a pre-processing mechanism called *inference materialisation*. This process computes the deductive closure of a KB and simply writes it back on disk or in memory. As a result, queries against the dataset only requires a simple look up, instead of complex inferences at query time. Materialisation is extremely powerful on static KBs because it only requires a costly phase once and for all, after which the system can sustain a heavy load of requests. If the KB changes sporadically, the closure can be recomputed entirely and the system can still perform well.

If the KB only changes monotonically, that is, nothing already known is ever removed or

*Corresponding author. E-mail: christophe.gravier@univ-st-etienne.fr

[1] In this paper, we assume a minimal familiarity with semantic web standards, especially RDF (the Resource Description Framework [20]), OWL (the Web Ontology Language [24]) and SPARQL (the query language for RDF [29])

[2] SPARQL 1.1 Update [11] became a recommendation in March 2013.

modified but new knowledge may be added, materialisation is also doable incrementally and can be efficient as well. Materialisation becomes costly whenever existing knowledge must be either modified or removed. In this paper, we are interested in a hybrid case, where new knowledge is simply added and materialisation enfolds, up until a conflict (that is, an inconsistency) occurs, which enforces the KB to be revised. This means to sort out obsolete assertions that were supported by the assertion that became obsolete.

Our contribution is a revision mechanism that is in line with the generic principles of belief revision [25,1], but offers a concrete instantiation that works for languages of the Semantic Web, namely OWL [**?**] and SWRL [10]. [3] Our approach is also constrained by certain requirements that come from an analysis of real use cases: (1) there is a fixed ontology that cannot be modified, however conflicting it may be with incoming knowledge; (2) the inferences made on the KB are materialised using a rule engine; (3) contextual knowledge are assertions of facts, in the description logic sense of the term, as opposed to terminological statements that are made in the fixed ontology; (4) we assume that the fixed ontology is consistent, so that consistency of the whole KB can always be guaranteed.

This paper is organised as follows. Section 1 describes the motivations that brought us to this specific problem that stems from smart environments. Section 5 provides related work with introductory definitions regarding knowledge revision. In Section 3.3, we present a justification-based update algorithm for conflicting OWL individual updates. Section 4 deals with experimentations and discussions of the proposal. Finally, Section 6 concludes.

1. Motivations

Conflicts are common place when knowledge comes from different context sources, provided by *ad hoc* services. Contextual data encompass network monitoring, user profiling, content descriptors and metadata, user-generated data, *etc.* [5]. In a situation where these contextual data are integrated, it is not possible to determine in advance which information should be simply added, and which trigger updates, that is, a modification of existing data. A system that collects such kind of information is what we call a context-aware system (CAS) because it often harvests raw data from sensors, such as the user's location, role, activity, device, connection, *etc.*, in order to customise the user's digital experience or notify different software agents [9].

We are especially interested in applications based on Semantic Web technologies, as these are gaining momentum in this field, and thus we consider *semantic context aware systems* (SCAS) to be the target of our contribution. At an abstract level, it means that we focus on formalisms like Description Logics and rule-based logics, which underly Semantic Web standards. At the concrete level, it means that we rely on a KB in OWL [24] and an associated reasoner [7], also combined here with rules in SWRL [10]. Using rules together with Description Logic ontologies is an effective technique for finding anomalies in ontologies [4], and especially in SCAS, where we want to notice undesired information and possibly correct them automatically.

We support the case when new assertions about the state of affairs are made and added to the system regularly, but sufficiently slow that it is possible to materialise inferences efficiently. We assume moreover, that incoming assertions are mostly providing more knowledge without contradicting existing facts, except occasionally such that inconsistencies can be handled sufficiently fast in average to avoid contradictions piling up. Note that the acceptable rate at which inconsistencies occur would strongly depend on the expressiveness of the ontologies and rules used. As we will see, our use case has a relatively complex ontology and rules, although we have a small number of terms.

The situation can be visualised in Figure 1. The picture shows that the SCAS is implemented in a software agent or a service [32]. The agent is maintaining a KB where the terminological knowledge is placed in a fixed ontology containing Description Logic axioms and rules. The fixed axioms can be seen as the physics of the KB, and nothing can supersede their truth. In addition to this, the agent has a dynamic set of assertions (the so-called A−Box in Description Logics) that is updated in

[3] Although SWRL is not a standard as OWL is, it is supported by several implementations and used in various applications.

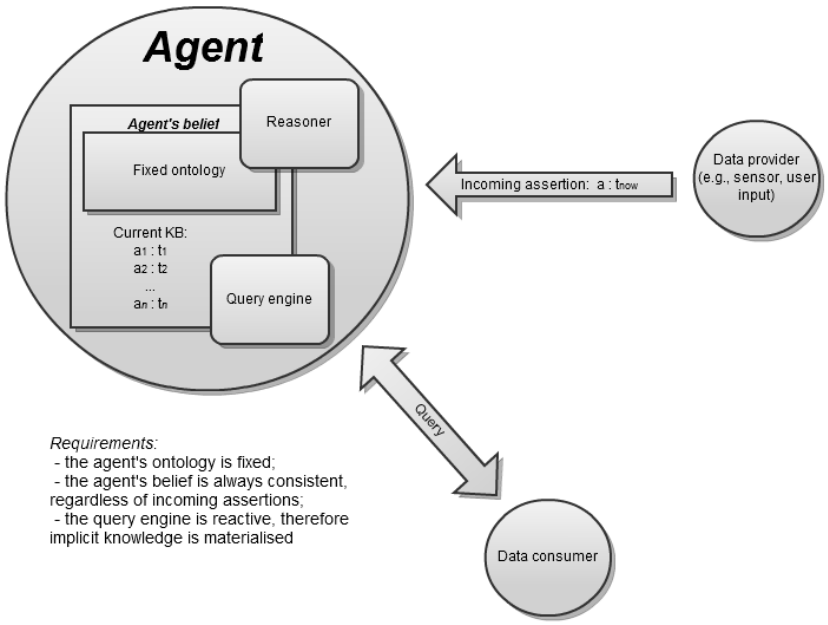

Fig. 1. Sematic context-aware service implemented in a agent or a service.

accordance with new assertions provided by the environment or other agents. A reasoner ensures that at all time, the A−Box contains the deductive closure, or at least relevant inferences are materialised. The query engine is at the service of the users and does not need carrying on reasoning at query time, because of the materialisation process. Therefore it can achieve a good level of reactiveness.

Repeatedly, the SCAS receives incoming assertions that complement its KB. In most cases, the assertions are simply added to the KB and the reasoner enfolds inferences. However, the assertions are accompanied with a context marker that will help the agent to revise knowledge appropriately in case of conflict. In particular, information such as who provided the new assertion (provenance), when the update occurred (timestamp), why it occurred (justification), *etc.* are all usable indicators for the revision strategy.

In this paper we investigate methods to retract the subset of materialised axioms that become obsolete as a result of the revision operations. This is crucial as removing a conflicting assertion may not be sufficient to cancel inconsistencies.

2. Reference scenario

In order to illustrate our work, we propose a reference scenario intended to be simple enough to serve as an illustration throughout the article, but having a certain degree of complexity to show that the approach is not limited to simplistic situations. The KB and the ontology that we consider covers an important DL fragment of OWL, in addition to including rules. We study the complexity and scalability of our proposal in Section 4.

Alice is reading at home in her room. We want to provide her a context-aware application that opens the windows of her room when the in-house temperature raises over 25.0°C. Meanwhile, would the temperature drop under a 15.0°C threshold, the system should close the windows.

The associated Semantic Context Aware System (SCAS) at her home held the following ontology $\mathcal{O}$, which can be expressed in description logics, with the usage of XML datatype built-in (boolean and float) for readability purpose, in Figure 2.

This ontology can be expressed in OWL [4].

[4]The associated OWL file can be browsed at `http://satin-ppl.telecom-st-etienne.fr/cgravier/jaise/cagc-test-jaise.owl`

T−Box**:**

$$\begin{aligned}
&\texttt{Window} && \equiv \leq 1\ \texttt{isOpened} \sqcap \forall \texttt{isOpened}.\text{boolean}\\
&\texttt{Room} && \equiv \exists \texttt{hasWindow} \sqcap \forall \texttt{hasWindow}.\texttt{Window}\\
&\texttt{CoolRoom} && \equiv \texttt{Room} \sqcap = 1\ \texttt{hasTemp} \sqcap \texttt{hasTemp}.\mathit{float}[\leq 15.0]\\
&\texttt{HotRoom} && \equiv \texttt{Room} \sqcap = 1\ \texttt{hasTemp} \sqcap \texttt{hasTemp}.\mathit{float}[\geq 25.0]\\
&\texttt{CoolRoom} && \sqsubseteq \neg \texttt{HotRoom}
\end{aligned}$$

A−Box**:**

$$\begin{aligned}
&\texttt{Room}(\texttt{AliceRoom})\\
&\texttt{Window}(\texttt{AliceRoomWindow})\\
&\texttt{hasWindow}(\texttt{AliceRoom}, \texttt{AliceRoomWindow})
\end{aligned}$$

Rules:

$$\begin{aligned}
&(R1)\ \texttt{CoolRoom}(?r) \wedge \texttt{Window}(?w) \wedge \texttt{hasWindow}(?r, ?w) \rightarrow \texttt{isOpened}(?w, \mathit{false})\\
&(R2)\ \ \texttt{HotRoom}(?r) \wedge \texttt{Window}(?w) \wedge \texttt{hasWindow}(?r, ?w) \rightarrow \texttt{isOpened}(?w, \mathit{true})
\end{aligned}$$

Fig. 2. Description of the ontology and rules used in our scenario.

Furthermore, Alice's room is equipped with a temperature sensor, whose samples lead to create the OWL assertions `HotRoom(AliceRoom)` or `CoolRoom(AliceRoom)`, depending if the sensor value reached any of the lower or upper thresholds for classification of the individuals as `CoolRoom` or `HotRoom`.

In order to provide this awareness mechanisms in a SCAS, thresholds are usually designed using rules. In our illustrative scenario, rules ($R1$) and ($R2$) can be serialised in SWRL [26,18].

Since context is volatile, OWL context individuals can radically change with time. For instance, let Alice's temperature sensor samples a new value equals to 25.0°C. The following assertion is inserted in the A−Box: `hasTemp(AliceRoom,`$25.0f$`)`. As a result, the assertion `HotRoom(AliceRoom)` is inferred by the reasoner, and in our case, materialised in the KB. Having `AliceRoom` being classified as a `HotRoom` by the reasoner has its consequences for the synchronised rule engine. It triggers the rule ($R2$), so that the assertion `isOpened(AliceRoomWindow,`*true*`)` is produced and is also added as an OWL assertion to the KB. This avoids to run again a potentially long classification, and helps in setting up a caching strategies to decreasing computation time for queries on the agent's KBs.

Later on, the same sensor issues a new value indicating that the temperature in Alice's room dropped to 14°C (it was hot inside, but opening the window cooled Alice's room). The assertion `hasTemp(AliceRoom,`$14.0f$`)` is added to the KB.

Obviously, the ontology $\mathcal{O}$ would fail a consistency check because there are two materialised values for the functional property *hasTemp*. In this case, we must remove the assertion `hasTemp(AliceRoom,`$25.0f$`)` from the ontology, but also the two assertions inferred from the old value of the temperature that are no longer supported: `HotRoom(AliceRoom)`, `isOpened(AliceRoomWindow,`*true*`)`.

The inconsistency is occurring because of a violation of a functional property axiom and because of the violation of a disjointness axiom. There are other situations that can lead to unsatisfiability, such as:

- the same keys (as in `hasKeys` axioms in OWL) being used for different individuals;
- a cardinality restriction violation (a generalisation of the violation of a functional property restriction);
- an individual relating to itself via a property that must be irreflexive;
- an asymmetric property used to relate individuals in both directions;

Other issues are raised due the shift of temperature. Not only should the oldest assertion about temperature be removed, but also all axioms that were materialized in previous infer-

ence because of this temperature, and that are only holding because of this now obsolete temperature. This includes in this example the assertion that Alice's room is classified as Hot (*HotRoom(AliceRoom)*) and that Alice's window should be open (*isOpened(AliceRoom, true)*).

A reasoner that embeds a justification mechanism can provide the precise reason why an inconsistency occurs, and we wish to exploit this feature in our revision process. We especially want to track down not only the assertion that is made directly obsolete by the new one, but also all the assertions that are made indirectly obsolete due to the new assertion. Otherwise, the ontology stays inconsistent.

3. Justifications-based resolution of conflicts on OWL Individuals updates

We present our approach by first defining some key concepts, then providing the actual algorithm for knowledge update.

3.1. Representation of context information

Since we expect that new coming information to be brought from the outside of the application (*e.g.*, a user update in a social platform or a sensor reading), and that it may be in conflict with already present knowledge, there must a revision strategy in place that crucially depends on the context of information. Most importantly, a SCAS must deal with possibly outdated knowledge being replaced by more recent one. Typically a sensor samples a new value that should update the assertion(s) associated with this context element (*e.g.*, the temperature in our reference scenario, speaking of the *hasTemp* functional property). In this reference scenario, the addition of such a new axiom leads to a clash due to consistency checking failure as illustrated at Section 1.

To keep our approach generic, we also envision the case when information is associated with a provenance, or with a confidence measure, or with credentials, and so on. Therefore, we need to represent contextual information in an abstract way, as an element of a set $\mathbf{C}$, where $\mathbf{C}$ is explicitly given on a per-application basis. In any case, there must exist a total ordering of $\mathbf{C}$. Given two elements of $\mathbf{C}$, our approach assume that it is always possible to identify the one that *supersede* (3.3) the other. For the purpose of our scenario, we assume $\mathbf{C}$ to be the set of timestamps. In a SCAS, all assertions are accompanied with an element of $\mathbf{C}$, serving as an indicator of the context of the assertion.

Definition 1 (Contextual assertion)**.** *A* contextual assertion *is composed of a logical assertion α and a context element $c \in \mathbf{C}$, noted α_c. For the sake of reasoning and semantics, a contextual assertion has the same meaning as the logical assertion that it contains.*

A contextual assertion α_c can also be read as "an occurrence of α in context c". In this work, a contextual assertions are either class assertion $C(a)$ or property assertion $R(a, b)$ or data property assertion $P(a, x)$ where C, R and P must be atomic. In our scenario, assertions are provided as OWL DL A−Box statements, and the context elements are time stamps, such as:

$$\texttt{hasTemp}(\texttt{AliceRoom}, 25.0f)_{\texttt{2012-12-12T12:12:12.12Z}}$$

The context element in a contextual assertion is momentous to the revision process, but it requires one additional constraint, namely that the set $\mathbf{C}$ be partially ordered. This is crucial in order to guide the strategy in identifying contextual assertions that are *preferred* over others. In our reference scenario, the ordering is given by chronology. Intuitively, this means that if t is a later instant than t', then α_t is supposedly "preferred" over $\alpha'_{t'}$. This reflects the fact that the most recent information is more reliable than older ones, in absence of other indicators. We will simply use the symbol '$<$' for the order relation, such that in our scenario, $t < t'$ means that t is an earlier time point than t'.

Other application scenarios can define it otherwise, but the overall approach and the algorithm we propose remain the same across all forms of contexts. Contextual assertions may be combined with terminological axioms or other assertions to draw new inferences. In this case, the inferences are not contextualised. Although it would be possible to provide a contextual reasoning framework (as in, *e.g.*, [33]), it is not our goal to exploit context for reasoning. We simply use it to guide the revision process, and as we will see later, it is important that we have a distinction between what is inferred and what is given as an assertion.

3.2. Justifications in OWL and conflicting OWL individuals updates

Our approach maintains a perpetually consistent KB. Consequently, if the reasoner associated with the system detects an inconsistency in a KB K, it means that it is the union of a new coming contextual assertion α_c with a consistent internal KB K. Formally:

$$K \cup \{\alpha_c\} \models \top \sqsubseteq \bot$$

The simplest approach for preserving consistency would be to simply reject α_c. However, in conformance with the postulates of belief revision (see Section 5), we consider that a new coming assertion is deemed true, or at least more reliable than earlier statements. As a result, it is required that the revision technique fetch one or several assertions from K in order to recover consistency. Again, to conform to the general belief revision postulates, we expect a minimal contraction of the ontology axioms.

In this work, we choose to base our approach on justifications, as proposed by [19,16]. A justification of an axiom for a KB is a minimal subset of the KB that entails the axiom. Formally:

Definition 2 (Justification)**.** *Let η be an axiom and K a KB. A* justification *of η with respect to K is a subset J of K such that $J \models \eta$ and for all $J' \subseteq K$, $J' \subset K \Rightarrow J' \not\models \eta$. We note $\mathcal{J}_K(\eta)$ the set of all justifications of η with respect to K.*

Note that if $K \not\models \eta$, the set of justifications $\mathcal{J}_K(\eta)$ is empty, otherwise it contains one or more justifications. As we explained in the motivation, we assume that the KB contains a fixed OWL ontology and the updates are always assertions relative to individuals, *i.e.*, A−Box assertions.

We are therefore interested in computing the justification of an inconsistency that arises from augmenting the current KB with such assertions. Assuming the current KB is K and the incoming assertion is α, inconsistencies occur when $K \cup \{\alpha\} \models \top \sqsubseteq \bot$. In order to restore consistency, as in belief revision, we want to explore the justifications of the inconsistent axiom, while preserving the incoming assertion. Intuitively, it appears to be obvious that computing the set $\mathcal{J}_{K\cup\{\alpha\}}(\top \sqsubseteq \bot)$ and removing an axiom or assertion from each justification would suffice as it guarantees recovering consistency. However, in the use cases we want to support, reasoning is performed by inference materialisations. This means that removing an axiom that previously produced materialised inferences should trigger the removal of said inferences.

As an example, let us assume that in our reference scenario the following contextual assertions are coming to the system, one after the other:

$$\texttt{hasTemp}(\texttt{AliceRoom}, 28.0f)_{\texttt{2012-12-12T12:00:00Z}}$$
$$\texttt{hasTemp}(\texttt{AliceRoom}, 13.0f)_{\texttt{2012-12-12T12:30:00Z}}$$

That is, the room was very hot at noon, and the window was opened, but the temperature drop fast, such that half an hour later, the room was pretty cold. This situation triggers an inconsistency in the KB, for which we have the following justifications:

$$\texttt{hasTemp}(\texttt{AliceRoom}, 28.0f)_{\texttt{2012-12-12T12:00:00Z}}$$
$$\texttt{hasTemp}(\texttt{AliceRoom}, 13.0f)_{\texttt{2012-12-12T12:30:00Z}}$$
$$\texttt{CoolRoom} \equiv \texttt{Room} \sqcap =1\,\texttt{hasTemp} \sqcap \texttt{hasTemp}.\mathit{float}[\leq 15.0]$$
$$\texttt{HotRoom} \equiv \texttt{Room} \sqcap =1\,\texttt{hasTemp} \sqcap \texttt{hasTemp}.\mathit{float}[\geq 25.0]$$
$$\texttt{CoolRoom} \sqsubseteq \neg\texttt{HotRoom}$$

There is a clear distinction between three groups of assertions and axioms that we rely on in our approach: the fixed terminological axioms (the T-Box axioms) denoted as A, the contextual assertions C and the inferred statements I, such that $K = A \cup C \cup I$. Our algorithm is doing two main operations:

- First, it selects assertions that should be discarded because they are sources of inconsistencies. This is done by considering all the justifications of the axiom $\top \sqsubseteq \bot$ in $T \cup C \cup \{\alpha\}$. Note that all such justifications necessarily contain α, as we assume that we start from a consistent K, and we exclude the inferred statements because the ones that could provoke an inconsistency are removed by the second phase. The choice of the assertion for each justification is based on the context marker associated with all assertions. More precisely, for a justification J, we choose an assertion $\eta_{c_{\min}} \in J$ such that $c_{\min} = \min(\kappa \in \mathbf{C} \mid \exists \beta_\kappa \in J)$. β_κ is an occurrence of β in context $\kappa \in \mathbf{C}$ (just like α_c was an occurence of α in context c earlier in this paper).

– Meanwhile, the assertions that are marked for deletion could have produced other assertions by way of inference materialisation. Since the deleted assertions are obsolete, so must be all the inferences drawn from them. Consequently, all assertions supported by assertions marked for deletion must also be marked for deletion. In this phase, only the inferred statements of I can be affected.

3.3. Algorithm

In order to keep track of what assertion supports an inferred one, we rely again on justifications. Note, too, that min is the minimal value according to the order on **C** as defined in Section 3.1. If the order is not

total, there may be several minimal elements, in which case we do not specify how to choose $c_{\min}$. Nonetheless, we identified three possible strategies:

– choose randomly one minimal element to discard;
– discard all minimal elements;
– use extra metadata to make the choice of $c_{\min}$ deterministic.

In any case, our reference scenario uses temporal instants that are totally ordered chronologically. Moreover, we also assumed in our motivations that the terminological axioms are fixed and should not be removed. This can be reflected by having a special context marker attached to T-Box axioms that is a top element of **C**.

In order to make concrete the algorithm informally presented in the previous section, we heavily rely on justifications. In fact, for all axioms and assertions in the KB, we compute their sets of justifications, keeping and exploiting only those that have a contextual assertion. This way, it is always possible to know the axioms and assertions that support a given statement, and the axioms and assertions that are supported by it.

To keep this section self-contained, we remind that the KB K has three distinct subsets A, C and I (fixed terminological axioms, contextual assertions and inferred statements), such that $K = A \cup C \cup I$ and $A \cup C \models I$. For ease of notations, we provide the following definition that we use later on:

Definition 3 (Supersede)**.** *Given an incoming assertion α and a justification J of the inconsistency of $A \cup C \cup \{\alpha\}$, we say that α* supersede *an assertion η_c with context marker c if and only if $c = \min(\kappa \in \mathbf{C} \mid \exists \beta_\kappa \in J)$. As a corollary, an obsolete axiom η_c marked with context marker c as an axiom being superseded by $\alpha, \alpha \in A \cup C$.*

Note that in the scenario we are considering, the concept marker is a time instant, so the superseded assertion is the one with the oldest time instant among the assertions justifying an inconsistency. We can nonetheless consider that the context marker ordering function can be user-defined, without modifying the genericity of our approach. For instance, other context marker include accuracy, trustworthiness, and other information of provenance information or correlation with other context information. The only requirement is to provide a totally ordered function for all context markers. In a nutshell, the algorithm can be presented as follows:

1. Add the new (contextual) assertion α to C.
2. Compute the sets of justifications for inconsistency of $A \cup C$, noted $\mathcal{J}_\perp$.
3. Build the sets of justifications $\tilde{\mathcal{J}}_\eta$ of η with respect to $A \cup C$ for each assertion η.
4. Start with an empty set of visited assertions.
5. For each justification $J \in \mathcal{J}_\perp$ do:
   (a) for each assertion $\eta \in J$ that α supersedes:
      – mark η as obsolete;
      – for all assertions κ that have justifications containing η do:
         * if all justifications in $\mathcal{J}_\kappa$ contain an obsolete assertion, mark κ as obsolete.
   (b) mark η as visited by the update algorithm.
6. Remove the obsolete assertions from K that were discovered by the update algorithm.

More formally this approach is provided as pseudo-code at Algorithm 1.

We explain the condition at Line 10: a fact β in the KB can only become obsolete if all of its justifications $J \in \mathcal{J}_\beta$ contain an obsolete fact γ. Otherwise, it means that the fact is also supported by other information that are not obsolete, and therefore the fact should remain.

input : A KB K and an incoming assertion α_c with context marker c
output: A revised consistent KB K^+ including α_c if possible, K otherwise.

begin
  $K^+ \leftarrow K \cup \{\alpha_c\}$
  Obsolete $\leftarrow \emptyset$
  Visited $\leftarrow \emptyset$
  $\mathcal{J}_\perp = \{J \subseteq K^+ \mid J \models \top \sqsubseteq \perp \wedge \forall J' \subset J, J \not\models \top \sqsubseteq \perp\}$
  for $J \in \mathcal{J}_\perp$ **do**
    for $\eta_d \in J$ *and* $\eta_d \notin$ *Visited and* α_c *supersedes* η_d **do**
      Obsolete $\leftarrow$ *Obsolete* $\cup \{\eta_d\}$
      for $\beta \in K^+$ *and* $\beta \notin$ *Obsolete* **do**
        if $\forall J \in \mathcal{J}_\beta, \exists \gamma \in \mathcal{J}_\beta, \gamma \in$ *Obsolete* **then**
          Obsolete $\leftarrow$ *Obsolete* $\cup \{\beta\}$
        end
      end
    end
    Visited $\leftarrow$ *Visited* $\cup \{\eta_d\}$
  end
  return $K^+ \setminus$ *Obsolete*
end

Algorithm 1: Revision, a justification-based A−Box update algorithm. The algorithm returns K^+, the minimally revised and consistent knowledge based after considering a new assertion α_c.

3.4. Claims and proofs about the update algorithm

In this section, we show that the algorithm is returning, as expected, a consistent ontology. We also show that, if some simple constraints are fulfilled, the algorithm behave as a belief revision algorithm. That is, given these constraints, an incoming assertion will be systematically included in the revised KB.

Theorem 1 (Total Correctness)**.** *Algorithm 1 terminates and returns a consistent KB.*

Proof. The termination of the algorithm is trivial since it iterates over finite sets. In the worst case, all assertions and axioms are visited, but the search eventually ends. The consistency of the returned KB is based on the fact that, if we remove one assertion or axiom from each justification, the resulting KB is necessarily consistent. The only issue that we have is that we do not allow our algorithm to remove axioms of the fixed ontology. So, for each justification, there must exist an assertion η_d that the incoming assertion α_c supersedes. For this to be possible, there must exist $d \leq c$ in the context markers of the assertions in the justification. But we have assumed that the KB K, prior to the insertion of α_c, is consistent. So in any case, the justifications of inconsistency always contain the assertion α_c. Therefore, if no other assertion exist in the justification, then at least α_c itself can be chosen as the superseded assertion. □

The possibility that an incoming assertion be rejected distinguishes our algorithm from a true belief revision mechanism. This can happen either if the context marker of the incoming assertion is lower than all the markers of other assertions involved in the inconsistency justification; or it can happen if the assertion is contradicting some of the ontology axioms. In the latter case, the new assertion is contradicting the internal model of the system and therefore should be eliminated. An example of such case happens with the assertion `ex:x owl:differentFrom ex:x`, which is in itself a contradiction and should be rejected. However, the following theorem shows that in relatively common situations, the algorithm behaves as expected.

Theorem 2. *If the following hold:*

1. *the fixed ontology A is a Description Logic ontology;*
2. *A only contains TBox axioms (by definitions);*
3. *the axioms in A do not contain nominals*[5] *or literal values (like integers, strings, etc.)";*
4. *all classes and properties are satisfiable*[6]*;*
5. *the context marker of the incoming assertion is higher (with respect to the order on contexts) than all the markers in the current KB;*

[5] A nominal is a class that is completely defined by its list of instances. It is usually written $\{a_1, \ldots, a_n\}$ in description logics syntax and means "the class that only contains the elements $a_1, \ldots, a_n$.

[6] A class is satisfiable if it can have at least an instance (*i.e.*, it is not necessarily empty). A property is satisfiable is there can be two elements in relationship with this property.

then Algorithm 1 returns a KB that contains the incoming assertion.

Proof. Let us consider the case where the assertion is of the form $C(a)$, and that the ontology is fulfilling the constraints given above. Since $C(a)$ has the highest context marker, it can only be rejected from the KB if it is the only assertion in a justification. Therefore, it would mean that $C(a)$ contradicts a subset of the ontology. But we have indicated that the ontology does not use nominals, so the symbol a never appears in the ontology. For this reason, from a logical perspective, the symbol a could be replaced by any symbol not occurring in the ontology without significantly changing the meaning of the statement. As a result, the instance a in the incoming assertion is in fact playing the same role as an existentially quantified variable that would have been skolemised, *i.e.*, it indicates that, in the justification, there exists some instance of class C. But by Item 4, this existence is guaranteed in all cases. Therefore, the assertion cannot contradict the ontology. □

Note that the constraint in Item 3 could be relaxed by saying that the instances appearing in the ontology are disjoint from the ones involved in the incoming assertions. Other sets of constraints could be devised for specific settings that would allow the same property to hold.

4. Evaluation

We shortly present our experimental setting before showing the results of our performance analysis and discuss about it.

4.1. Implementation

We have implemented this algorithm using the ontology of the reference scenario exposed in Section 2. In the implementation of the reference use case of this paper, context changes are synthetised by producing A−Box assertions that conform to the ontology we have introduced. Initially, Alice's room has a temperature of 26.0°C, corresponding to the assertion `hasTemp(AliceRoom, 26.0`f`)`. The reasoner classifies `AliceRoom` as a `HotRoom`, and its windows are opened. The two following assertions are therefore materialised: `HotRoom(AliceRoom)` and the assertion `isOpened(AliceRoomWindow, true)`. Then a new assertion is added to the KB simulating that the temperature of Alice's room dropped to 14°C at the next context sampling (therefore `hasTemp(AliceRoom,14.0`f`)` is added). At that stage, the reasoner's KB does not pass the consistency check. Our algorithm automatically triggers and revises both the obsolete context assertions and inferred knowledge from these, so that the reasoner's KB is updated to reach a consistent state. This leads the machine to remove the following assertions:

- `HotRoom(AliceRoom)`
- `hasTemp(AliceRoom,` $14.0f$`)`
- `isOpened(AliceRoom, true)`

The revised ontology proved to be consistent against Pellet reasoner consistency checking.

4.2. How to reproduce the experiment

The ontology can be retrieved online[7], along with the source code of our reference scenario running the algorithm[8]. It is written in Java, using OWL API version 3.1.0 and the Pellet Reasoner from Clark & Parsia[9], in version 2.2.2.

4.3. Performance analysis

From Section 3.3, two major operations are subject of performance analysis:

- The computation of inconsistency (as provided by [16])
- The update of OWL assertions to preserve consistency (including consistency checking). We split this into two sub algorithms:
  - the main algorithm of our OWL assertions update process as described at Algorithm 1
  - the subroutine that recursively finds all obsolete OWL individuals because they are directly or indirectly supported by the new assertion that causes a clash.

[7] `http://satin-ppl.telecom-st-etienne.fr/cgravier/jaise/cagc-test-jaise.owl`

[8] `http://satin-ppl.telecom-st-etienne.fr/cgravier/jaise/CACGExample.java`

[9] `http://clarkparsia.com/`

We ran all our experiments on a laptop equipped with a 2.4 GHz Intel Core 2 Duo processor and 4 Gb of DDR2 SDRAM.

The use case provided in Section 4.1 took 907 ms (in average over 50 runs) to execute the entire process: loading the ontology in memory, performing a consistency check, adding a new assertion that provokes a clash, checking consistency again, which fails, and up to the update and recursive algorithm provided in this paper, and the final consistency check of the revised ontology that is successful.

The distribution among the algorithms is as follows: the computation of inconsistency took 5 ms (0.5 % of total execution time), the update algorithm took 831 ms (91.6 %).

In order to get a proper idea of each of the three major algorithms performance and time of execution depending on the number of assertions in the ontology, we decided to set a performance analysis based on the number of OWL assertions in both A−Box and T−Box. We implemented a generator that expands the OWL ontology provided by the use case supporting this paper. The generator increase the number of elements in the A-Box ("Individuals" axix in Figure 3) and on the T-Box ("Concept" axis on the 3). For each possible number of elements in A-Box and T-Box between 0 and 200, we ran our experiment when provoking a conflict (hence 4,000 runs). For these numbers, the algorithm exhibits an interesting linear complexity. We have not tested for higher values as we want to address lightweight ontologies, especially when embeded within agents (Figure 1).

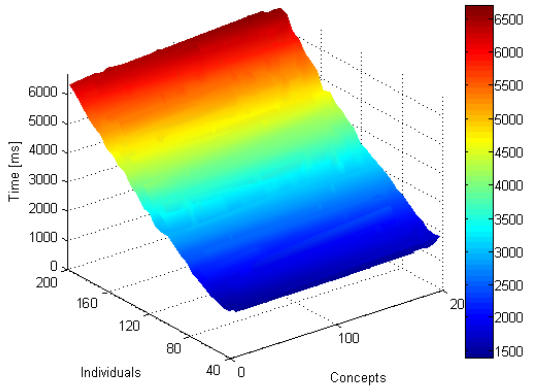

Fig. 3. Computation time for different A−Box and T−Box size

4.4. Discussion and further improvement

The higher expressiveness that OWL offers over RDFS allows one to express much richer representations of situations, environments and contexts, both digital and physical [30]. However, it comes at a computational cost that calls for trade-offs when using OWL in pervasive computing or under other dynamic conditions. While the current trend in Semantic Web ontology engineering is to keep ontologies as simple as possible [27], there are circumstances under which it is necessarily—or at least desirable—to have soundness, completeness and expressiveness at the same time.

Such an approach is not meant to support large scale ontologies [10]. Instead, it fits smaller KBs, as in a multi-agent distributed environments, where each agent holds a reasonable KB size. The common approach is to build peer-to-peer infrastructure enriched with simple distributed ontologies in order to guarantee good computational properties (as in [28,3]). Moreover, would real-time justification-based updates be unaccessible for now for medium to large ontologies, it is still able to address such cases asynchronously as the algorithm terminates (*cf.* Proof 3.4).

Furthermore, while we assume the fixed ontology to be rather small, it can still generate a large amount of assertional facts, as a result of materialisation. Therefore, strategies based on re-evaluation from scratch may not be as powerful in practice, as it breaks the whole purpose of pre-computing inferences for quick responses.

Our algorithm does not specify how to compute the justifications and we simply reused an existing implementation for OWL DL ontologies. This means that consistency has to be checked each time a new assertion is made, in additional to all the reasoning for materialisation. We could couple consistency checking with inference materialisation to significantly speed up the process.

We also notice that we do not yet have a minimal contraction of the knowledge base. This is due to the strategy we apply for choosing which assertion to remove. As we iterate through the justifications for inconsistencies, we may mark for deletion an assertion that also appear in a different

[10] Yet recent research show that given some theoretical guarantees on the characteristics of ontologies, it may be possible to offer both scalability and expressiveness [17]

justification. In the latter justification, a different assertion could be marked for deletion, because there can be an older context attached to it. This happened very little in our experiments, but we will investigate solutions to this issue.

5. Related Works

Our proposed solution is guided by pragmatic requirements that bring together many research fields, in particular belief revision and paraconsistent reasoning, Semantic Web data store and reasoner implementation, inference justifications. To the best of our knowledge, no existing proposal addresses the problem as described at Section . Nonetheless, similar problems were addressed in different fields. These fields are belief revision, Semantic Web query Languages, triplestore implementations, and defeasible reasoning. In this Section we discuss what were these propositions, and how they fall short in completely addressing our problem.

First of all, our approach implements a mechanism that is strongly based on the general principles of belief revision in [25,1]. We nonetheless diverge a bit from the original postulates of belief revision, as we will describe in Section 3. The postulates of a revision operation + are the following: [11]

Closure $K + \phi = \mathsf{Cn}(K + \phi)$
Success $\phi \in K + \phi$
Inclusion $K + \phi \subseteq K \cup \{\phi\}$
Vacuity if $\neg\phi \notin K$, then $K + \phi = K \cup \{\phi\}$
Consistency $K + \phi$ is consistent if ϕ is consistent
Extensionality if $\phi \Leftrightarrow \psi$, then $K + \phi = K + \psi$

In the foundational definition of [1], K is assumed to be a closed belief set, but this assumption has been criticised from a conceptual and technical point of view. In this paper, we are more interested in belief *base* revision rather than belief *set* revision. Belief bases are finite sets of formulas and therefore, the first postulate of belief revision does not hold for belief bases. Vacuity has to be changed to "if $K \not\models \neg\phi, then K + \phi = K \cup \{\phi\}$". In our paper, we are using a weaker notion of the success postulate, that [15] call *relative success*, namely, "$\phi \in K + \phi$ or $K + \phi = K$".

In [21], it has been shown that belief revision can be reduced to the notion of circumscription, introduced in [23]. This way, they prove that algorithm for circumscription can be used to achieve belief revision. Circumscription is formalising the fact that "everything should be as expected unless stated otherwise". In that sense, it defines a formalism similar to default logic. Previous attempts to introduce the notion of circumscription in OWL [12] were made but there is no support for this kind of formalism in today's state of the art OWL engines.

Adding statements and maintaining consistency by retracting other statements may be done using the standard Semantic Web query language SPARQL with its update feature [11]. However, this standard alone can only ensure that specifically chosen statements are removed from the database, which does not guarantee by itself that other inferred statements do not generate inconsistencies. Even with SPARQL Update, it is still necessary to keep track of the statements that must be retracted.

Existing triple store implementations do not have built-in features for maintaining consistency. The only triple stores that match our motivating scenario are those that materialise inferred triples from given ones, like AllegroGraph [12] or OWLIM [13]. In these stores, inferred statements are separated from other statements, such that if a modification occurs, especially with a DELETE query, the inferred statements can be recomputed from the explicit statements. However, those triple stores are simply regenerating all the inferences from the new dataset, while they could only remove the statements that are not anymore supported by the deleted statements. Under the non-monotonicity paradigm, different approaches tried to address the update of knowledge in DLs (defeasible reasoning). Among the existing works in this field, we have come to classify them into two classes. The first category promote a strategy where some knowledge has the priority over other, such that to avoid inconsistencies. The second class of approaches tracks changes inside the reasoner to maintain its state.

[11] In the postulates, K identifies a belief set, *i.e.*, a set of closed logical formulas, ϕ and ψ are formulas, and Cn is the closure operation.

[12] `http://www.franz.com/agraph/allegrograph/`
[13] `http://www.ontotext.com/owlim`

The first class includes defeasible rules and defeasible reasoning, such as in [2,13]. Especially, [13] introduces an annotation approach for prioritising rules, in order to prevent contradiction clashes. All these approaches are based on the strategy to prioritise some selected knowledge over other pieces of knowledge, either by ordering the statements by introducing an overriding mechanism for rules. Another kind of similar approach is paraconsistent reasoning, which aims at reasoning in presence of inconsistency [22].

A globally different strategy was employed by algorithms that maintain the internal state of DL reasoners. The commonly implemented strategy is to keep trace in a graph of the precedence (and succedence) relationships between axioms for each incremental inference. Upon update, the graph is pruned starting from the node representing the axiom that became obsolete [25,14]. These works follow the associated proposals in DLs from [31] (dependency direct backtracking), and [8] (data dependency network management). Unfortunately, the few implementations in a OWL DL reasoner do not cope with inconsistency that may occur when expanding or contracting the KB [14]. Moreover, all these works serve the purpose of maintaining the in-memory model of the reasoner, rather than keeping the consistency of a queryable knowledge base.

Finally, we mention the work on justification as it is part of the core of our proposed algorithm. A justification of a logical formula with respect to a KB is a minimal subset of the KB that entails the formula. Justifications have been studied and implemented for Semantic Web languages, and are mainly used to debug ontologies (*e.g.*, in the Protégé 4 ontology editor). However, justificactions can have various other applications as we show in Section 3.

6. Conclusion

The dynamics of semantic web applications or semantic-web-based systems, together with the need for reactivity, offer a challenge to SCASs that are based on ontologies and rules. With our proposed approach, we expect to address the dynamics with an update and revision mechanism that takes care of inconsistencies, while keeping a fair level of reactivity with a materialisation process. We assume that an open environment with unpredictable inputs such as a sensor network or social platform needs a system that tolerates any generated facts that do not contradict foundational knowledge and common rules. Yet, it must be able to provoke inconsistencies by setting precise and expressive rules and axioms in a well defined ontology.

This goes against a more traditional approach where a specific business logic is applied in order to programme what can be updated and how it can be updated. The drawbacks of those approaches are the lack of flexibility, durability and the difficulty to interoperate in a distributed, open environment.

In this paper, we presented an automatic approach to deal with conflicting OWL individuals updates. These operations are transparent for the context-aware system developer. We used justifications of a consistency check clash in order to automatise the process of minimising the KB contraction that occur due to the need to maintain its consistency. The proposed algorithm may go as far as revising the context KB in order to maintain its consistency. This approach had been implemented using Pellet reasoner and the OWL API.

Still issues arise, such as the optimisation of the algorithm, its distribution over several software agents and how to merge provenance models and update justifications in order to customise OWL individual updates strategies developed by different semantic-empowered software agents.

Acknowledgments

This work was supported by Conseil Général de la Loire and by the project OpenCloudware (`http://opencloudware.org`), which is funded by the French Fonds national pour la Société Numérique (FSN), and is supported by Pôles Minalogic, Systematic and SCS. The authors also want to thank the reviewers for their careful reading of the paper and their helpful comments.

References

[1] C. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of symbolic logic*, 50:510—530, Jan 1985.

[2] G. Antoniou and A. Bikakis. DR-Prolog: A system for defeasible reasoning with rules and ontologies on the semantic web. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):233–245, Jan 2006.

[3] F. Baader, C. Lutz, and A.-Y. Turhan. Small is again beautiful in description logics. *KI-Künstliche Intelligenz*, 24(1):25–33, Jan 2010.

[4] J. Baumeister and D. Seipel. Anomalies in ontologies with rules. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(1):55–68, 2010.

[5] Y.-G. Billet, C. Gravier, and J. Fayolle. Context-awareness for Next-Generation Applications Servers. *MRC, CONTEXT'11, 26th-30th September, Karlsruhe, Germany*, 2011.

[6] Y.-G. Billet, C. Gravier, and J. Fayolle. Swrl-based context awareness for application servers hosting digital services. In *Rule-Based Modeling and Computing on the Semantic Web*, pages 222–229. Springer Berlin Heidelberg, 2011.

[7] Y.-G. Billet, C. Gravier, and J. Fayolle. SWRL-based context awareness for application servers hosting digital services. *RuleML2011BRF, 3rd-5th November 2011, Fort Lauderdale, Florida, USA*, 2011.

[8] E. Charniak, C. Riesbeck, D. McDermott, and J. Meehan. Artificial intelligence programming. *Hillsdale NJ: Lawrence Erlbaum Associates*, Jan 1980.

[9] S. Dourlens, A. Ramdane-Cherif, and E. Monacelli. Tangible ambient intelligence with semantic agents in daily activities. *JAISE*, 5(4):351–368, 2013.

[10] I. H. et al. SWRL: A Semantic Web Rule Language - Combining OWL and RuleML. W3C member submission, May 21 2004.

[11] P. Gearon, A. Passant, and A. Polleres. SPARQL 1.1 Update - W3C Working Draft 5 January 2012. W3C Working Draft, Jan. 5 2012.

[12] S. Grimm and P. Hitzler. Defeasible Inference with Circumscribed OWL Ontologies. *Workshop on Advancing Reasoning on the Web: Scalability and Commonsense*, 2008.

[13] B. Grosof, M. Dean, and M. Kife. The SILK System: Scalable Higher-Order Defeasible Rules. *RuleML, LNCS, Springer, Heidelberg*, 5858, Jan 2009.

[14] C. Halashek-Wiener, B. Parsia, and E. Sirin. Description logic reasoning with syntactic updates. *5th Int'l Conf. on Ontologies, Databases, and Applications of Semantics (ODBASE 2006), LNCS, Berlin, Heidelberg: Springer-Verlag*, 4275(722-737), Jan 2006.

[15] S. O. Hansson, E. L. Fermé, J. Cantwell, and M. A. Falappa. Credibility limited revision. *J. Symb. Log.*, 66(4):1581–1596, 2001.

[16] M. Horridge, B. Parsia, and U. Sattler. Laconic and precise justifications in OWL. *ISWC 2008, LNCS, Springer*, 5318:323–338, Jan 2008.

[17] I. Horrocks. Semantics $\sqcap$ scalability $\models \bot$? *Journal of Zhejiang University - Science C*, 13(4):241–244, 2012.

[18] I. Horrocks, P. F. Patel-Schneider, and S. Bechhofer. OWL rules: A proposal and prototype implementation. *Web Semantics: Science and Services and Agents on the World Wide Web, Elsevier Ed.*, 3(1):23—40, Jan 2005.

[19] A. Kalyanpur, B. Parsia, and E. Sirin. Debugging unsatisfiable classes in OWL ontologies. *Web Semantics: Science and Services and Agents on the World Wide Web, Elsevier Ed.*, 3(4):268–293, Jan 2005.

[20] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax W3C Recommendation 10 February 2004. W3C Recommendation, Feb. 2004.

[21] P. Liberatore and M. Schaerf. Reducing belief revision to circumscription (and vice versa). *Artificial intelligence*, 93:261—296, Jan 1997.

[22] Y. Ma, H. Hitzler, and L. Zuoquan. Paraconsistent Reasoning for Expressive and Tractable Description Logics. *Franz Baader, Carsten Lutz, Boris Motik, Proceedings of the 21st International Workshop on Description Logics, DL2008, Dresden, Germany, May 2008. CEUR Workshop*, 353, 2008.

[23] J. McCarthy. Circumscription–a form of non-monotonic reasoning. *Artificial intelligence*, 13:27—39, Jan 1980.

[24] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview W3C Recommendation 10 February 2004. W3C Recommendation, Feb. 10 2004.

[25] B. Nebel. Reasoning and revision in hybrid representation systems. *LNAI, Springer-Verlag, Heidelberg*, 422(1-6), Jan 1990.

[26] M. J. O'Connor, H. Knublauch, S. Tu, B. Grosof, M. Dean, W. Grosso, and M. Musen. Supporting rule system interoperability on the semantic web with SWRL. *ISWC'05, LNCS, Springer*, 3729:974–986, Jan 2005.

[27] D. Preuveneers and P. Novais. A survey of software engineering best practices for the development of smart applications in ambient intelligence. *Journal of Ambient Intelligence and Smart Environments*, 4(3):149–162, Aug. 2012.

[28] M.-C. Rousset. Small can be beautiful in the semantic web. *ISWC 2004*, Jan 2004.

[29] A. Seaborne and E. Prud'hommeaux. SPARQL Query Language for RDF. W3C Recommendation, Jan. 15 2008.

[30] T. Springer and A.-Y. Turhan. Employing description logics in ambient intelligence for modeling and reasoning about complex situations. *Journal of Ambient Intelligence and Smart Environments*, 1(3):235–259, Aug. 2009.

[31] R. M. Stallman and G. J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial intelligence*, pages 135—196, Jan 1977.

[32] H. J. ter Horst and A. Sinitsyn. Structuring reasoning for interpretation of sensor data in home-based health and well-being monitoring applications. *Journal of Ambient Intelligence and Smart Environments*, 4(5):461–476, 2012.

[33] A. Zimmermann, N. Lopes, A. Polleres, and U. Straccia. 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.