Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Subset Reasoning for Event-Based Systems

PIETER BONTE, FEMKE ONGENAE, AND FILIP DE TURCK.
DLab, Department of Information Technology at Ghent University - imec, Technologiepark 126 Ghent, 9052, Belgium
Corresponding author: Pieter Bonte (e-mail: pieter.bonte@ugent.be).

ABSTRACT In highly dynamic domains such as the Internet of Things (IoT), Smart Industries, Smart Manufacturing, Pervasive Health or Social Media, data is being continuously generated. By combining this generated data with background knowledge and performing expressive reasoning upon this combination, meaningful decisions can be made. Furthermore, this continuously generated data typically originates from multiple heterogeneous sources. Ontologies are ideal for modeling the domain and facilitates the integration of heterogeneous produced data with background knowledge. Furthermore, expressive ontology reasoning allows to infer implicit facts and enables intelligent decision making. The data produced in these domains is often volatile. Time-critical systems, such as IoT Nurse Call systems, require timely processing of the produced IoT data. However, there is still a mismatch between volatile data and expressive ontology reasoning, since the incoming data frequency is often higher than the reasoning time. For this reason, we present an approximation technique that allows to extract a subset of data to speed-up the reasoning process. We demonstrate this technique in a Nurse Call proof of concept where the locations of the nurses are tracked and the most suited nurse is selected when the patient launches a call and in an extension of an existing benchmark. We managed to speed up the reasoning process up to 10 times for small datasets and up to more than 1000 times for large datasets.

INDEX TERMS Reasoning, Streams, Event-Based, Ontology, Nurse Call

I. INTRODUCTION

A. PROBLEM DESCRIPTION

Highly dynamic domains such as the Internet of Things (IoT), Smart Industries, Smart Manufacturing, Pervasive Health, financial sector or Social Media require real-time processing of heterogeneous generated data [1]. These time-critical systems need to react as quick as possible to newly generated event data. However, many of these systems need to integrate background knowledge with the event data on the fly, to enable real-time interpretation of these events and execute advanced logics to make correct decisions [2], [3]. For instance, in an IoT nurse call system, expressive reasoning is required to capture the capabilities of the nurse, the pathologies of the patients, the relation between the patients and the staff, etc [4]. To automatically determine the priority of a launched patient call, the pathology of the patient needs to be inspected. Depending on the patient's disease, the call gets a higher priority. Similar examples can be thought of in other domains such as detecting hazard situations in a smart manufacturing scenario or reacting to traffic jams in smart cities.

Semantic web technologies, such as ontologies, are the preferred model for the integration of the generated heterogeneous data with background knowledge [5], [6]. An ontology formally describes concepts, properties, and their relations, within a certain domain. By defining the relations between various concepts, a model can incorporate the knowledge about a certain domain. Through the use of a reasoner, implicit facts can be automatically inferred. For example, by modeling that a 'high priority call' is a call made by a patient that has a certain risk profile, the reasoner can automatically decide which calls should be handled with higher priority. Note that the fact that a patient has a risk profile can be inferred based on the pathology and the history of the patient. To make intelligent conclusions, the reasoner should be able to handle highly expressive definitions in the ontology. We opted for Web Ontology Language (OWL) reasoning, which uses Description Logic (DL), as OWL is widely used and it is a web standard.

However, currently, there is still a mismatch between expressive reasoning and real-time requirements [1]. Expressive reasoning techniques such as DL reasoning can have up to NEXPTIME complexity [7], resulting in slow reasoning times with growing datasets [8], [9]. In this paper, we present a practical subset reasoning technique that combines expressive reasoning and event-based requirements by extracting and approximating a subset of data. The subset minimizes that dataset to reason upon, speeding up the reasoning process.

Many advances have been made in the Stream Reasoning domain [10], [11], [12] to combine data from multiple streams with static background knowledge. Generated event data produced by various sources can be considered as data streams. To be able to process these unbounded streams of data, stream reasoning techniques consider the data within a defined time frame, i.e., a window. Expressive reasoning platforms have mostly focused on the processing of static data [13] or slowly changing data [14]. When these systems try to process data streams, newly incoming data will pile up, eventually crashing the system, since each item needs to be processed one by one [15]. By using windows, multiple data items can be processed simultaneously. However, a problem that arises when using windows in combination with expressive reasoning, is the possible inconsistency within a window. For example, when an individual is a member of two disjunct classes due to considering the data within the window. However, the content of the window should never be inconsistent, only the most recent statement should be considered [16]. Handling these inconsistencies within a window is still an open problem [16].

B. RELATED WORK

We now discuss the most prominent works in the literature and their drawbacks.

Traditional OWL2 DL reasoner such as HermiT [13] and Pellet [17] focus on the processing of static or very slow changing data. They provide no mechanisms for the processing of event data and are typically too slow to handle event data.

PAGOdA [18] is a hybrid approach that combines a datalog reasoner with an OWL2 reasoner. Most of the computations are executed by the fast datalog reasoner and only if necessary the OWL2 reasoner computes the missing facts. Although it is a very promising technique, in its current state PAGOdA focuses on querying and does not allow the adding and removal of facts which is necessary in a changing environment. This means that the PAGOdA reasoner would need to be restarted each time new data arrives. It is built for static data and does not support reasoning over data streams. PAGOdA uses RDFox [14] as its datalog reasoner. RDFox is the fastest incremental OWL2 RL reasoner currently available. As we will discuss in Section III, OWL2 has three profiles that minimize the expressivity to increase the efficiency of reasoning. RDFox is thus not as expressive as OWL2 DL reasoning. Furthermore, as PAGOdA and RDFox focus more on static domains, they do not provide any mechanisms to process data streams, such as windowing or update policies.

TrOWL [19] offers a subset of OWL2 DL expressiveness while maintaining tractable, by using language transformations. It supports stream reasoning by incrementally processing the addition and removal of facts. As only a subset of OWL2 DL is supported, TrOWL does not support nominals or datatype reasoning.

Many RDF Stream Processing (RSP) techniques exist [11], [10] that consider streams of RDF data within a predefined window and process only the content within the window. When new data arrives, or when time passes, the window slides over the data stream and a new portion of the data stream is processed. These techniques support the integration with background data and support various streaming operators such as aggregations. However, due to the high velocity of data these systems need to process, the reasoning capabilities are low. The most expressive RSP engine is StreamQR [20], which supports the $\mathcal{ELHIO}$ logic, which falls under the OWL2 EL fragment, one of the most expressive logics currently used for query rewriting. As OWL2 EL is another profile of OWL2 DL, it is less expressive.

Other techniques such as module extraction and ontology partitioning focus on minimizing the ontology Terminological Box (TBox). Module extraction techniques [21] allow to extract a part of the TBox to speed up the reasoning process in a specific case, e.g. to type check some specific classes. While ontology partitioning techniques split the ontology into smaller self-contained modules [22]. Both techniques focus on minimizing the TBox but provide no solution for growing Assertion Box (ABox)es. Anagnostopoulos et al. [23] highlight the importance of approximate reasoning in order to perform time-critical decision making. They utilize probabilistics to approximate certain reasoning tasks, based on the similarity with other situations, without dealing with highly expressive ontologies. This implies that the results might not always be correct. In our approximation approach, we aim for correct answers achieved by approximating a subset of data to perform the expressive reasoning upon.

C. OBJECTIVES & SOLUTION

To allow the design of time-critical systems within highly dynamic domains, we set the following objectives:

1) *Heterogeneous data:* Since data typically needs to be combined from various heterogeneous sources, we need to be able to integrate this heterogeneous data.
2) *Event data:* Data is continuously produced, therefore new data should be added to the system and old data should be updated or removed.
3) *Large knowledge bases:* Many domains have large knowledge bases that need to be combined with the generated data, e.g. sensors typically only describe their sensor readings and still need to be combined with the sensor itself and the location of the sensor, etc.
4) *Expressive reasoning :* In order to correctly interpret the domain, expressive reasoning is required to correctly

analyze the domain definitions.

To tackle these challenges, of performing expressive reasoning over event data, we propose an approximation technique that minimizes the dataset to reason upon, in order to speed up the reasoning process in an event-based environment. To solve the windowing problem, we define various update policies that describe how the most recent data in the stream should be captured. As such, instead of a window, we maintain a recent view on the stream and use this recent view to compute our subset.

We show that our subset approach scales very well, even with large amounts of data (i.e. instantiation data), making it an ideal tool for time-critical systems that require expressive reasoning.

D. PAPER ORGANIZATION

The remainder of this paper is structured as follows: Section II introduces the nurse call use case used throughout the paper. In Section III the background to understand the remainder of the paper is explained. Section IV describes the policies that allow to maintain a recent view on the data stream. Section **??** explains the subset approximation algorithm. While Section VI describes the implementation details of the system. Section VII evaluates our technique by comparing the execution time of the use case from Section II and a benchmark to existing techniques and discusses the results. We show that our technique is up to 10 times faster for really small datasets and up to more than 1000 times faster for larger datasets. Section VIII discusses how the results should be interpreted and Section IX concludes the paper and identifies interesting paths for future work.

II. USE CASE DESCRIPTION

In the remainder of the paper, we focus on an IoT nurse call system to introduce and explain our approach. We note that our approach is applicable for any event-based environment requiring expressive reasoning.

The IoT plays a crucial role in providing optimal and personalized care for patients. The advances in this field allow patients to be easily monitored [24] and to localize the necessary staff members [25]. However, to achieve truly personalized care, profile, context and domain information needs to be considered. Most of this information is rather static, e.g., the patient's profile and pathology, the competences of the nurses and the floor-plan of the hospital. Discrete streams of events representing, e.g. patient' calls, person location updates, call status updates, need to be combined with the static data to derive actionable insights. In this use case, we consider a call assignment scenario, where the most suited staff member at a particular moment should be assigned to a patient call. The nurse selection procedure is made up out of a rather large decision tree consisting of 36 leaves [4]. The tree was constructed together with domain experts, i.e. nurses, doctors and patients, and a company specialized in nurse call systems (Televic Healthcare[1]). The selection procedure takes into account, amongst others, the personal relation of the staff members with the patients, the location of the staff members and their competences. We consider the following scenario within this paper, which typically occurs during the night, consisting of the following steps:

1) **Call Launched:** A patient launches a call and the selection procedure is started to assign the most appropriate nurse to the call. The nurse is notified of this assignment.
2) **Call Redirect:** The nurse is currently busy and indicates that the call should be redirected. The selection algorithm runs again to assign and notify another nurse.
3) **Call Temporary Accept:** The new nurse temporary accepts the call. This is a temporary accept because the call can only be completely accepted once the nurse is with the patient. This allows easy re-assignments in case of interruptions or delays.
4) **Corridor:** The nurse moves towards the room of the patient and continuous location updates are registered by the IoT system.
5) **Patient Location:** When the nurse arrives in the patient room, a new location update is sent. Some lights in the room automatically turn on at the appropriate low level, since a staff member is present in the room.
6) **Presence on:** The nurse logs into the terminal in the patient room. The call is now accepted and the correct lights, depending on the procedure, turn on. For example, for a medical procedure, the spotlights above the bed turn on, while for an assistance procedure the mood lighting is activated.
7) **Presence off:** The nurse inputs some additional administrative information about the procedure of the call on the terminal and logs out. The call is now finished and the procedure lights turn off.
8) **Corridor:** The nurse leaves the room. The location of the nurse is updated and since no staff member is in the room, all the lights turn off.

Since regulations stipulate that a nurse should be present in the room within three or five minutes (depending on the country) when a call has been made, the allowed decision time should be limited to five seconds, to allow for plenty of time for the nurse to move to the room. Therefore, the data should be processed in a timely manner to meet these real-time requirements. To represent the eHealth knowledge, the ACCIO ontology[2] is used, which has been constructed in collaboration with domain experts. An elaborate description can be found in Ongenae, et al. [4].

The decision tree of the selection procedure was translated into SPARQL queries, that takes into account the background, profile and context information captured within the ACCIO ontology.

[1]http://www.televic-healthcare.com/
[2]https://github.com/IBCNServices/Accio-Ontology/

III. BACKGROUND

This section introduces the necessary background on which the remainder of this paper is built.

A. DESCRIPTION LOGICS

The popularity of OWL has led to the design of OWL2, defining the foundations of OWL2 DL reasoning [26]. **Description Logics** [27] are the logical-based formalisms on which OWL2 DL has been built [28]. We introduce the syntax of a simplified DL, explaining the basic notions to understand the remainder of the paper. We refer the reader to Horrocks et. al. [29] for a more thorough description of the logic $\mathcal{SROIQ}$ (which is used within OWL2 DL) and its semantics. DL defines *concepts* to represent the classes of individuals and *roles* to represent binary relations between the individuals. Concrete roles (or data properties) are roles with datatype literals in the second argument.

DL languages contain *concepts names* $A_1, A_2, ...,$ *role names* $P1, P2, ...$ and *individual names* $a_1, a_2, ...$ A *role* R is either a role name P_i, its inverse P_i^- or a complex role $R_1 \circ \cdots \circ R_n$ consisting of a chain of roles. *Concepts* C are constructed from: two special primitive concepts $\bot$ (bottom) and $\top$ (top) or concept names and roles using the following grammar:

$$C ::= A_i|\top|\bot|\neg C|C_1 \sqcap C_2|C_1 \sqcup C_2|\exists R_1.C_1|\forall R_1.C_1$$

Note that the two last concepts are called, respectively existential ($\exists$) and universal ($\forall$) quantifiers. More expressive constructs such as qualified number restrictions are allowed as well:

$$C ::= \geq nR.C_1| \leq nR.C_1$$

Meaning that at least or at most a specific number n of relations R should be present. Nominal support allows to restrict to specific individuals instead of concepts:

$$C ::= \exists R.\{a\}|\exists R.\{a_1, a_2, ..a_n\}$$

Where the latter can be seen as "one-of". Data property restrictions can restrict the values of data properties:

$$C ::= \exists R. \geq n|\exists R. \leq n$$

A TBox $\mathcal{T}$, is a finite set of concept (C) and role (R) inclusion axioms of the form

$$C_1 \sqsubseteq C_2 \text{ and } R_1 \sqsubseteq R_2$$

with C_1, C_2 concepts and R_1, R_2 roles. A concept equation ($C_1 \equiv C_2$) denotes that both C_1 and C_2 include each other:

$$C_1 \sqsubseteq C_2 \text{ and } C_2 \sqsubseteq C_1$$

An ABox $\mathcal{A}$ is a finite set of concept and role assertions of the form

$$C(a) \text{ and } R(a, b)$$

with C a concept, R a role and a and b individual names. $ind(\mathcal{A})$ denotes the set of individuals occurring in $\mathcal{A}$. A *Knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ combines $\mathcal{T}$ and $\mathcal{A}$.

We can now define the semantics using an interpretation $\mathcal{I}$. $\mathcal{I}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non-empty domain of interpretation $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$. The interpretation function assigns:

- an element $a_i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to each individual name a_i,
- a subset $A_i^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to each concept name A_i,
- a binary relation $P_i^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each role name P_i.

We can now use the interpretation function to define the semantics of the above defined grammar:

$$\begin{aligned}
(P^-)^{\mathcal{I}} &= \{(v, u)|(u, v) \in P^{\mathcal{I}}\},\\
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}},\\
\bot^{\mathcal{I}} &= \emptyset,\\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}},\\
(C_1 \sqcap C_2)^{\mathcal{I}} &= (C_1)^{\mathcal{I}} \cap (C_2)^{\mathcal{I}},\\
(C_1 \sqcup C_2)^{\mathcal{I}} &= (C_1)^{\mathcal{I}} \cup (C_2)^{\mathcal{I}},\\
(\exists R_1.C_1)^{\mathcal{I}} &= \{u|\exists v \in C^{\mathcal{I}} \wedge (u, v) \in R^{\mathcal{I}}\},\\
(\forall R_1.C_1)^{\mathcal{I}} &= \{u|\forall v \in C^{\mathcal{I}} \wedge (u, v) \in R^{\mathcal{I}}\}.
\end{aligned}$$

We call $\mathcal{M} = \mathcal{K}^{\infty}$ the **materialization** of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, i.e. all inferred axioms w.r.t. explicit individuals in $\mathcal{K}$ are computed and explicitly stored. For example, based on the knowledge defined in $\mathcal{T}$, additional axioms regarding $\mathcal{A}$ can be extracted.

OWL2 contains three profiles, each limiting the expressivity power in a different way, to ensure efficiency of reasoning:

- OWL2 RL: does not allow existential quantifiers on the right-hand side of the concept inclusion, eliminating the need to reason about individuals that are not explicitly present in the knowledge base. Furthermore, it does not allow quantified restriction, e.g. minimum, maximum or exactly a specific number of quantified roles. This profile is ideal to be executed on a rule-engine.
- OWL2 EL: mainly provides support for conjunctions and existential quantifiers. This profile is ideal for reasoning over large TBoxes that do not contain, among others, universal quantifiers, quantified restrictions or inverse object properties.
- OWL2 QL: does not allow, among others, existential quantifiers to a class expression or a data range on the left-hand side of the concept inclusion. This makes the profile ideal for query rewriting techniques.

Note that each of these profiles is a subset of OWL2 DL. Expressive logics such as OWL2 DL require special techniques to support their reasoning, such as the tableaux algorithm [30], [31], i.e. a proof mechanism for first-order logic, which is provided by reasoning systems [17], [13], [32], [33].

B. REASONING METHODS

Looking at the literature, three categories of reasoning methods can be distinguished:

- *Reason at query time:* These approaches perform the reasoning while executing the query. The query evalu-

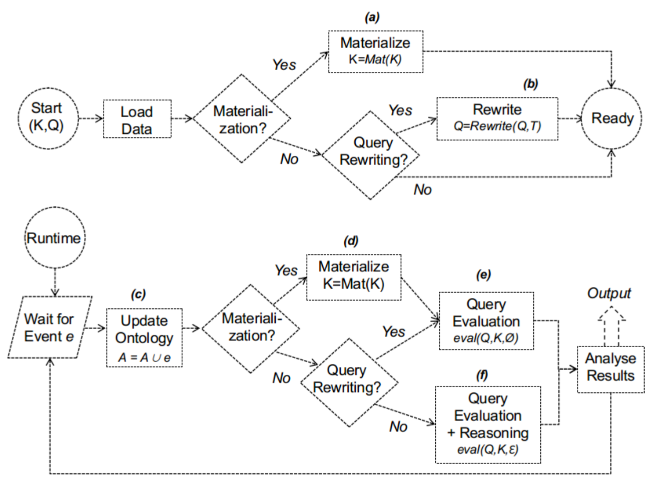

FIGURE 1. Visualization of the flow of materialization, query rewriting and reasoning at query time approaches.

ation itself is then performed under a certain entailment, in order to incorporate the reasoning. The query evaluation can be formalized as $eval(Q, \mathcal{K}, \mathcal{E})$, with Q the query that needs to be evaluated, $\mathcal{K}$ the ontology ABox and TBox and $\mathcal{E}$ the entailment regime that defines the expressivity of the reasoning during query evaluation.

- *Materialization:* Materialization approaches materialize their data, such that queries can be executed without the need for reasoning during query execution. The query evaluation can be formalized as $eval(Q, \mathcal{K}^\infty, \emptyset)$, with $\mathcal{K}^\infty$ the materialization of $\mathcal{K}$. The entailment regime is $\emptyset$, denoting that there should be no reasoning while evaluating the query.
- *Query Rewriting:* These approaches rewrite the provided queries based on the ontology TBox, such that the query contains all the information from the ontology. The query evaluation can here be defined as $eval(Q', \mathcal{K}, \emptyset)$, with Q' the rewritten query such that the reasoning is contained within the query. The entailment regime is here also $\emptyset$, thus no reasoning should be executed while evaluating the query. We note that only a subset of OWL2 DL can be rewritten.

Figure 1 visualizes the flow of the different reasoning methods. We make a distinction between the loading of the data at start-up, at the top, and the adding of event data during runtime, at the bottom. The figure shows that, once the event data has been added to the ontology at runtime, the rewriting and materialization approaches don't require reasoning at query time. However, the materialization results in computing all possible assertions in the knowledge base, some might be irrelevant for the query answering. While the query rewriting results typically in a very complex query and only a small subset of OWL2 DL can be rewritten.

IV. UPDATE POLICIES

In an event-based environment, data is continuously produced and special techniques are necessary to capture the current view on the data streams. We introduce the definition of *Update Policies* that allow to define how the current view should be constructed. The current view differs from the window operator as it is updated based on the previous current view, while the window is an operator that captures the stream in processable chunks. We start with the definition of a current view on a data stream:

Definition 4.1: Ax is a set of ABox axioms and $S = Ax_1, Ax_2, ..., Ax_n$ is a time varying sequence of axioms. $Ax_S(t)$ is a function that extracts a view at a specific time instant from the Stream S. $Ax_{current}$ is a **current view** on an axiom Stream S, it describes as a set of axioms how the updates in the stream should be interpreted at the current time.

We can now introduce the notion of an update policy:

Definition 4.2: An **Update Policy** U_p is a function that updates the view with the incoming axioms of the stream to a current view:
$Ax_{current}(t) = U_p(Ax_{current}(t-1), Ax_S(t))$, with t the latest time instant.
Let's say we have a stream S of axiom sets Ax_i, i.e. $S = Ax_1, Ax_2, ..., Ax_n$ with $i = 1..n$ time instances.

We define three basic update policies. Figure 2 visualizes the differences between these policies based on different axioms set in the data stream.

1) **Latest:** always takes the latest axiom set in the data stream, similar to the *Now* operator that can be defined for window functions in streams, namely:
$$U_P Latest(Ax_1, Ax_2) = Ax_2$$
2) **Combine:** merges the sets of axiom's together.
$$U_P Combine(Ax_1, Ax_2) = Ax_1 \cup Ax_2$$
3) **Update:** overwrites existing relations and merges new ones.
$$U_P Update(Ax_1, Ax_2) = (Ax_1 \backslash Ax^-(Ax_1, Ax_2)) \cup Ax_2$$
with
$$Ax^-(Ax_1, Ax_2) = \{R(a,b) \in Ax_1 | \exists R(a,c) \in Ax_2 \wedge b \neq c\}$$

Note that more advanced policies are possible, however, out of scope for this paper.

V. SUBSET REASONING

Now that we can maintain a view on the data stream according to the defined policy updates, we introduce the concept of subset reasoning as an approximation technique to efficiently speed up the reasoning process. The technique calculates a subset of ABox data from the knowledge base, based on the new axioms in the data stream. The extracted subset allows to efficiently calculate the materialization of the new axioms. Note that this requires the knowledge base to be in a materialized state.

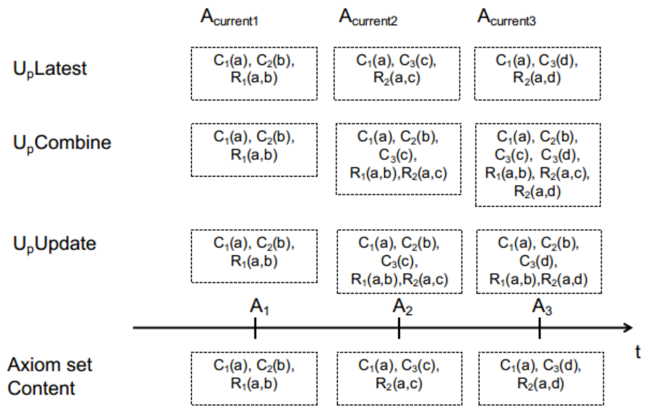

FIGURE 2. Visualization of resulting current views based on the different update policies.

A. DEFINING SUBSET REASONING

Before we define how to extract the subset, we need to identify how big the subset should be (similar to the Modal depth [34]):

Definition 5.1: We define the **Concept Depth** as a recursive function, starting from all axioms $C_{eq} \in \mathcal{T}$ with C_{eq} a concept inclusion or concept equivalence and C_i concepts and R_i roles:

$$\begin{aligned}
depth(C_1 \sqsubseteq C_2) &= max(depth(C_1), depth(C_2)) \\
depth(C_1 \equiv C_2) &= max(depth(C_1), depth(C_2)) \\
depth(C_i) &= 0 \text{ with } C_i \text{ a concept name} \\
depth(C_1 \sqcap C_2) &= max(depth(C_1), depth(C_2)) \\
depth(C_1 \sqcup C_2) &= max(depth(C_1), depth(C_2)) \\
depth(\exists R.C_1) &= depth(R) + depth(C_1) \\
depth(\forall R.C_1) &= depth(R) + depth(C_1) \\
depth(R_1 o ... o R_n) &= depth(R_1) + ... + depth(R_n) \\
depth(R_i) &= 1
\end{aligned}$$

The Concept depth thus calculates the number of relations defined in a concept. We can now calculate the deepest concept within the TBox:

Definition 5.2: We define the **TBox Depth** as:

$$depth(\mathcal{T}) = max(\{depth(C) | C \in \mathcal{T}\}).$$

The TBox depth will define how big the subset should be.

Example 5.1: We calculate the TBox depth based on the following example TBox:

$$\begin{aligned}
\mathcal{T} = \{ & \\
& NormalCall \equiv Call \sqcap \exists callMadeBy. \\
& \qquad \exists hasRole.(Patient \sqcup Resident) \\
& CareCall \equiv NormalCall \sqcap \exists hasReason.CareReason \\
& PriorityCall \equiv Call \sqcap \exists callMadeBy. \\
& \qquad \exists hasProfile.RiskProfile \\
& Patient \equiv Role \sqcap \exists hasDetails. \\
& \qquad \exists isAdmittedTo.Hospital \\
& Resident \equiv Role \sqcap \exists hasDetails. \\
& \qquad \exists isAdmittedTo.ResidentCareCenter \\
\} &
\end{aligned}$$

The depths for each of these concepts is respectively:

$$\begin{aligned}
depth(NormalCall) &= 2 \\
depth(CareCall) &= 1 \\
depth(PriorityCall) &= 2 \\
depth(Patient) &= 2 \\
depth(Resident) &= 2
\end{aligned}$$

The TBox depth is $max(\{2, 1, 1, 2, 2\}) = 2$.

Before introducing the definition of a subset, we define the collection of all outgoing relations as the function r_{out}:

Definition 5.3: The outgoing relations of an individual i in a knowledge base $\mathcal{K}$ is defined as:

$$r_{out}(i, \mathcal{K}) = \{R(i, j) | R(i, j) \in \mathcal{A}\}$$

The types of the individuals that are linked through these outgoing relations are defined through the function c_{out}:

$$c_{out}(i, \mathcal{K}) = \{C(j) | R(i, j) \in r_{out}(i, \mathcal{K}) \wedge C(j) \in \mathcal{A}\}$$

The combination of all the outgoing relations with the types of the linked individuals is then defined as rc_{out}:

$$rc_{out}(i, \mathcal{K}) = r_{out}(i, \mathcal{K}) \cup c_{out}(i, \mathcal{K})$$

We can now define how a subset for a set of ABox axioms Ax can be calculated with respect to a materialized knowledge base $\mathcal{K}^\infty$. We denoted $ind(Ax)$ as the collection of individuals contained in Ax and $depth_{max}$ the TBox depth.

Definition 5.4:

A subset $Sub(Ax, \mathcal{K}^\infty, depth_{max}) = \{R_{out}(i, \mathcal{K}^\infty, depth_{max}) | i \in ind(Ax)\}$ with

$$\begin{aligned}
R_{out}(i, \mathcal{K}, 0) =& rc_{out}(i, \mathcal{K}), \\
R_{out}(i, \mathcal{K}, depth) =& rc_{out}(i, \mathcal{K}) \\
& \cup \{R_{out}(i', \mathcal{K}, depth - 1) | i' \in ind(r_{out}(i, \mathcal{K}))\}
\end{aligned}$$

So the subset method of an ABox Ax based on $\mathcal{K}^\infty$ extracts recursively all relations, individuals and their types linked to the individuals in Ax that are also in $\mathcal{K}^\infty$. The recursion is dependent on the TBox depth. Note that to make the theory work in practice two special cases need to be incorporated:

- When in the latest step of the scenario (depth=0) there are transitive relations, the recursion should continue to follow these relations (and these relations only) until no more of the transitive relations are found.
- In each step, we store the ABox relations that have been followed before and make sure we do not follow previously visited relations. This is to prevent that the algorithm gets stuck in a loop.

These special cases were not included in the formalization in order to maintain clarity.

Example 5.2: (cont'd) We extend the TBox from Example 5.1 with the following axioms:

$$\begin{aligned}\mathcal{T}' = \mathcal{T} \cup \{&BehaviourRiskProfile \sqsubseteq RiskProfile\\ &MedicalRiskProfile \sqsubseteq RiskProfile\}\end{aligned}$$

We introduce the following ABox:

$$\begin{aligned}\mathcal{A}' = \{&Person(p1), hasRole(p1, r1), Role(r1),\\ &Hospital(h1), hasProfile(p1, m1),\\ &MedicalRiskProfile(m1), Detail(d1)\\ &hasDetails(r1, d1), isAdmittedTo(d1, h1),\\ &\ldots\\ &Person(p2), Person(p3), Person(p4),\\ &hasRole(p2, r2), hasRole(pr, r3), hasrole(p4, r4),\\ &StaffMember(r2), CareRole(r3),\\ &StaffMember(r4)\\ &\}\end{aligned}$$

Note that $\mathcal{A}'$ contains more axioms (indicated by '...'), but these were omitted for conciseness reasons. When we materialize the knowledge base, we can infer (among others) that $Patient(r1)$ and $RiskProfile(m1)$. Consider now the following update in the stream:

$$Ax = \{Call(c1), callMadeBy(c1, p1)\}$$

If we calculate the subset for Ax in function of $\mathcal{T}'$ and $\mathcal{A}'$, we obtain the following axioms:

$$\begin{aligned}&Sub(Ax, \mathcal{K}^\infty, 2)\\ &\quad= \{R_{out}(c1, \mathcal{K}^\infty, 2), R_{out}(p1, \mathcal{K}^\infty, 2)\}\\ &\quad= \{Call(c1), callMadeBy(c1, p1), Person(p1),\\ &\qquad hasRole(p1, r1), R_{out}(r1, \mathcal{K}^\infty, 1), R_{out}(m1, \mathcal{K}^\infty, 1)\}\\ &\quad= \{Call(c1), callMadeBy(c1, p1), Person(p1),\\ &\qquad hasRole(p1, r1), Role(r1), hasDetails(r1, d1),\\ &\qquad R_{out}(d1, \mathcal{K}^\infty, 0), hasProfile(p1, m1),\\ &\qquad MedicalProfile(m1), RiskProfile(m1)\}\\ &\quad= \{Call(c1), callMadeBy(c1, p1), Person(p1),\\ &\qquad hasRole(p1, r1), Role(r1), hasDetails(r1, d1),\\ &\qquad Detail(d1), hasProfile(p1, m1),\\ &\qquad MedicalProfile(m1), RiskProfile(m1)\}\end{aligned}$$

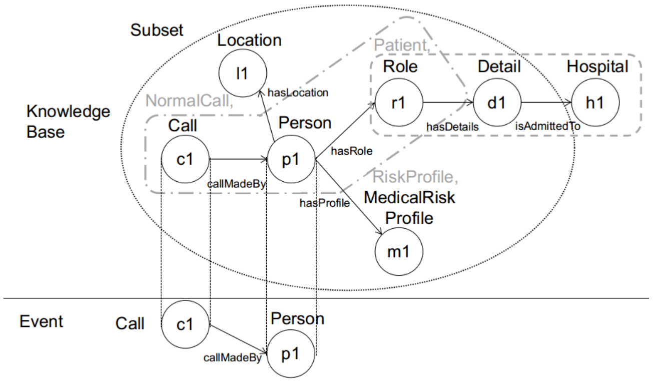

FIGURE 3. Visualization of the example illustrating that subset technique can extract a subset of data and still infer the types of the individuals in the event data due to the materialization of the knowledge base. The circle visualizes the data in the subset, the dotted rectangles depict the data necessary to infer that c1 is a NormalCall and r1 is a Patient.

Which is a subset and still allows us to calculate the fact that $c1$ is a $NormalCall$ and a $PriorityCall$.

The example is visualized in Figure 3 where we can see that we don't need all information in the knowledge base to infer the types of $c1$. To infer that the Call $c1$ is a NormalCall, we need to know that $r1$ is a Patient. However, the subset does not contain all the data to infer that $r1$ is a Patient. Since we operate on a materialized knowledge base, we already inferred in a previous step that $r1$ is, in fact, a Patient. Since the inferred types are part of the subset, we can successfully infer that $c1$ is a NormalCall. Since the knowledge base is materialized, a smaller set of data can be extracted to infer the types of the data in Ax.

B. SUBSET REASONING: A PRACTICAL APPROACH TOWARDS EXPRESSIVE EVENT-BASED REASONING

The subsetting technique is very useful upon frequently changing data. We define an axiom streaming dataset as follows:

Definition 5.5: An **axiom streaming dataset** ADS is a set of the form $\{(A_S, U_p), \ldots\}$ with A_S a stream of events described as axioms and U_P its update policy, as defined in Section IV. In an event-based environment such as the IoT, there are multiple streams to be considered since data from various sources needs to be combined. The calculated subsets for each of these streams should thus be combined with the knowledge base to allow querying of the integrated streams. The process of the various steps for one stream is depicted in Figure 4. First, we calculate the new current view through the selected policy update and based on the new view we extract a subset of data from the materialized knowledge base. The subset is then materialized and combined with the materialized knowledge base in the Subset knowledge base. The Subset knowledge base can then be queried.

Definition 5.6: A **Subset Knowledge Base** is the union of the materialized knowledge base and the materialized axioms sets from ADS: $\mathcal{K}^\infty_{ADS} = \bigcup A^\infty_S \cup \mathcal{K}^\infty$ with $A^\infty_S =$

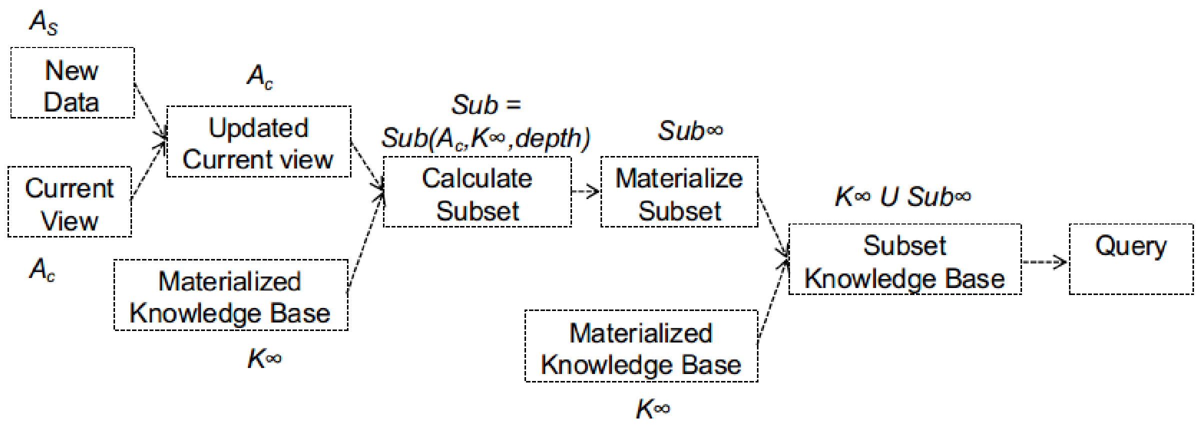

FIGURE 4. The different processing steps of the practical subset reasoning approach. With A_c the current view ($A_{current}$) on the stream A_S.

$\mathcal{M}(Sub(U_P(A_S current, A_S(t)), \mathcal{K}^\infty, depth, \mathcal{T}))$

This allows to combine the materialized views on the different streams with the static data in the knowledge base. For each A_S we use a subset of $\mathcal{K}^\infty$ to materialize $A_S current$. Due to the monotonicity property, $\mathcal{K}^\infty$ does not need to be updated when data is removed from $A_S current$ since the materialization of $A_S current$ is maintained outside of $\mathcal{K}^\infty$. Additional information regarding $\mathcal{K}^\infty$ can be inferred in $A_S current$ but it does not inflict $\mathcal{K}^\infty$ directly. Since the union of $\mathcal{K}^\infty$ and all the $A_S current \in ADS$ are used for query answering, these results are taken into account. Note that when $A_S current$ is updated, the previous materialization is removed and a new materialization based on the extracted subset is calculated. This methodology bypasses the difficulties attached to removals in an incremental reasoning approach [35], [36] (i.e. detecting which inferred facts should be removed when removing data).

C. EXTENSION OF SUBSET REASONING

Since the subsetting technique is an approximation technique it is not always complete. It is however always sound. The technique focusses on the efficient materialization of the new events, however, updating the knowledge base according to the new events might sometimes be incomplete if these updates fall outside of the subset. We first introduce a pure fictional scenario to highlight the possible risks and afterwards we present a solution. Note that in the practical implementation of the nurse call system, we were never confronted with these limitations. Figure 5 visualizes these limitations. Let's consider the following TBox consisting of wards, patients and calls that might be at risk:

$$\begin{aligned} RiskWard &\equiv Ward \sqcup \exists hasPatient.RiskPatient \\ RiskPatient &\equiv Patient \sqcup \exists madeCall.RiskCall \\ RiskCall &\equiv Call \sqcup \exists hasStatus.RiskStatus \\ callMadeBy &\equiv madeCall^- \end{aligned}$$

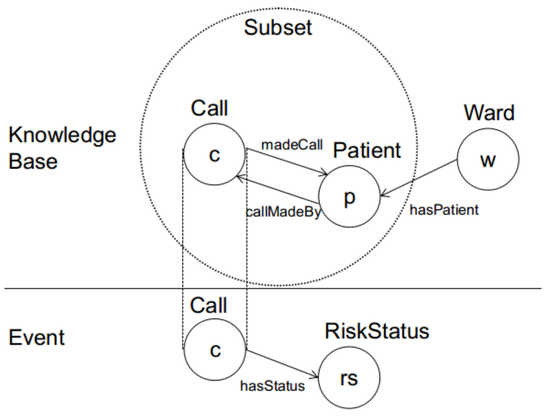

FIGURE 5. Visualization of the limitations of the subset technique when the events update parts of the knowledge base outside the subset.

Say that the knowledge base consists of the following ABox:

$$Ward(w), Patient(p), hasPatient(w, p),$$
$$Call(c), madeCall(p, c), callMadeBy(c, p)$$

The stream that updates the call statuses A_S produces the following axioms that state that the call made by the patient has a risk status:

$$Call(c), hasStatus(c, rs), RiskStatus(rs)$$

Since the TBox depth is only one in this example, we extract the following subset:

$$Call(c), hasStatus(c, rs), RiskStatus(rs),$$
$$callMadeBy(c, p), Patient(p)$$

Upon materialization of the subset based on the introduced TBox, we can infer that that call c is a RiskCall and that patient p is a RiskPatient.

$$RiskCall(c), RiskPatient(p)$$

However, since the ward w was not in the subset, we do not infer that w is a RiskWard.

$$RiskWard(w)$$

As a solution to this problem, we can gradually increase the subset in size when individuals at the edge of the subset have changed type. With the edge of the subset, we denote the individuals that are present in the subset but are not in A_S.

$$edge(A_S, \mathcal{K}^\infty, d) = \{c|c \in (ind(Sub(A_S, \mathcal{K}^\infty, d)) \backslash ind(A_S))\}$$

The set of individuals that changed type in the edge of the subset can be captured by the following function:

$$\begin{aligned} edge_{Changes}(A_S, \mathcal{K}^\infty, d) = \\ \{c|\exists c \in edge(A_S, \mathcal{K}^\infty, d) \wedge \exists C \in \mathcal{T} \\ \wedge\, C(c) \in Sub^\infty(A_S, \mathcal{K}^\infty, d) \\ \wedge\, C(c) \notin Sub(A_S, \mathcal{K}^\infty, d)\} \end{aligned}$$

With Sub^∞ the materialization of the subset. In the case that the changes A_c in the edge (i.e. $A_c = edge_{Changes}(A_S, \mathcal{K}^\infty, depth_{max})$) is not empty, we extend the subset by adding the changes A_c to the ABox we want to calculate the subset upon:

$$Sub(A_S \cup A_c, \mathcal{K}^\infty, depth_{max})$$

We keep increasing the subset ABox, with the changes A_c, until there are no more individuals at the edge that have changed type, i.e. until $A_c = \emptyset$. By also including changed individuals in the edge in the calculation of the subset, individuals that should be influenced by these changes will also be correctly materialized.

Furthermore, when calculating the subset, we also take the incoming relations r_{in} and their types rc_{in} into account in this extension.

$$\begin{aligned} r_{in}(i, \mathcal{K}) =& \{R(j, i)|\exists R \in \mathcal{T} : R(j, i) \in \mathcal{A}\} \\ c_{in}(i, \mathcal{K}) =& \{C(j)|\exists R(j, i) \in r_{in}(i, \mathcal{K}) : C(j) \in \mathcal{A}\} \\ rc_{in}(i, \mathcal{K}) =& r_{in}(i, \mathcal{K}) \cup c_{in}(i, \mathcal{K}) \end{aligned}$$

When we redefine rc_{out} used in the subset extraction function, then these relations are also taken into account:

$$rc_{out}(i, \mathcal{K}) = r_{out}(i, \mathcal{K}) \cup c_{out}(i, \mathcal{K}) \cup rc_{in}(i, \mathcal{K})$$

Note that the incoming relations in the original subset approach are not necessary since there we only want to infer the types of the newly added data and its closest relations.

In the example, Patient p became a RiskPatient and p was in the edge of the subset. Therefore we add p to the arriving data and calculate the subset on their union. This results in the following subset:

$$Call(c), hasStatus(c, rs), RiskStatus(r, s), \\ callMadeBy(c, p), Patient(p), Ward(w), hasPatient(w, p)$$

This subset is sufficient to infer that w is a RiskWard. We note that in most of our scenarios it is the knowledge base that

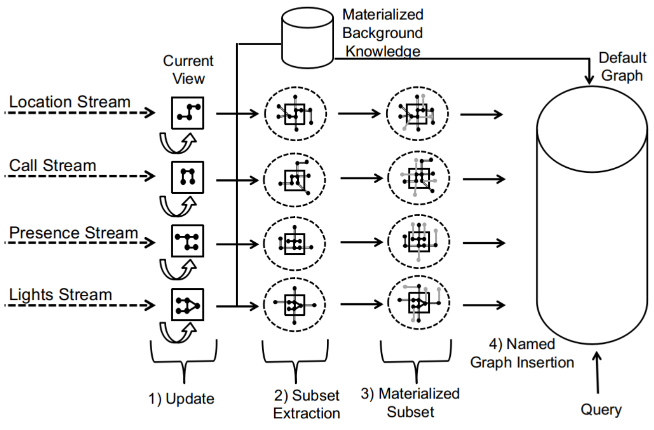

FIGURE 6. The different processing steps of the subset reasoning approach.

influences the arriving events and its nearest relations instead of the other way around. Therefore, we propose the latter solution as an optional configuration as it needs to verify the types of the individuals at the edge and thus slightly slows down the process.

VI. REALIZING SUBSET REASONING

In this section, we describe how we practically implemented the subset reasoning approach[3]. Figure 6 visualizes the various steps.

For simplicity, we first assume that different streams of observations can be distinguished from each other. For example, in our use case, we have a location stream that transmits the observations regarding the new locations of the personnel, a call stream regarding new calls and their updates, a light stream that captures the current status of lights and a presence stream that indicates whether the personnel logs in on the various devices. If these streams cannot be distinguished, additional filtering, such as defined in Section VI-E, can further split the streams up. The streams can be modeled as OWLAPI[4] axioms or Jena[5] statements. Furthermore, in the below explanation we assume that the background knowledge is already materialized in a preprocessing step and the queries are already defined, such that they can be continuously executed. If this is not the case, we also provide the mechanisms to materialize an ontology.

A. UPDATING

As each stream produces data, the stream updates are first fed through their own policy updater that calculates the current view on the stream according to the policies defined in Section IV. This is visualized in Figure 6 as step 1. The system maintains a current view for each stream and when new data arrives the current view is updated according to the

[3]The implementation itself is available on github.com/pbonte/SubsetReasoning

[4]http://owlapi.sourceforge.net/

[5]https://jena.apache.org/

policy. In Example 5.2 we already showed an update in the call stream.

$$\mathcal{A}_{\mathcal{S}} = \{Call(c1), callMadeBy(c1, p1)\}$$

Since this is the beginning of a call, the current view will be empty and the whole update will be considered the new current view.

One can register a new stream by indicating the stream name and the policy update for that stream. When adding new data, the name of the stream should also be passed as an argument. This allows to extract the correct current view of the stream and update it according to the defined update policy.

B. SUBSET EXTRACTION

Once the current view on the stream has been fixed, the subset can be calculated from all the individuals in the current view and the (materialized) background knowledge. This is done according to the description in Section **??** and visualized in Figure 6.2. We refer the reader to Example 5.2 for an example of the extracted subset based on the above defined current view.

C. MATERIALIZING SUBSET

Once the subset is extracted, the whole subset is materialized according to the used ontology TBox (Figure 6.3). An example of the materialization of the extracted subset can be found in Example 5.2. The reasoning is performed with the Hermit reasoner. However, other reasoners could easily be plugged in.

D. NAMED GRAPH INSERTION

Each of the materialized subsets is then added as a named graph to an RDF store. The background knowledge details the default graph. Upon querying, the various graphs are merged and the querying is done on the union of all the graphs. We utilized the jena Dataset to store the various graphs and to query them.

E. FINE GRAINED FILTERING

As one can imagine, the performance of the system strongly depends on the size of the subset that needs to be materialized. It is possible that the arriving streams produce observations that could be further split up. For example, in our hospital setting, the location updates could come from multiple wards, while it is only necessary to combine updates from the same ward. Therefore we support finer-grained filtering of the observations to split them up according to some parameter, e.g. a ward in the hospital. This is done by allowing additional SPARQL queries that select that parameter. To do this filtering, the stream observation needs to be combined with the materialized background knowledge.

Example 6.1: To filter each location update according to the ward it is produced in, we combine the produced data, containing the observation, with the knowledge base and execute the following query that selects the ward:

```
SELECT DISTINCT ?ward WHERE {
GRAPH :location {?s ?p ?o}
?s context:hasLocation ?loc.
?loc context:hasCentreCoordinate ?coord.
?coord context:hasZCoordinate ?ward.
}
```

The ward level is then used as an additional identifier to select the current view of the location stream for that specific ward.

F. DIFFERENCES TRADITIONAL REASONING METHODS

Looking back at the reasoning methods introduced in Section III-B, i.e. reasoning at query time, materialization and query rewriting, Subset reasoning is clearly a materialization approach. However, it differs in the way the materialization is calculated and maintained. In Figure 1 we have seen that data is typically just added to the ontology (step c), then the knowledge base is materialized (step d) and then the querying can be executed (step e). Our Subset reasoner takes a different approach where the updates to the ontology are stream specific and can overwrite and update data through the use of the update policies. Thus step c of Figure 1 aligns with Figure 6 step 1 The materialization of the runtime data is also specific for each stream and utilizes the subset extraction to minimize the data to reason upon. Thus step d of Figure 1 aligns with both Figure 1 step 2 and step 3. The querying itself requires an additional step, as first all the materializations of all the streams need to be combined with the materialized background knowledge. By extracting only a subset of data, the reasoning process can be speed up.

G. SUBSET REASONING CONFIGURATION

The Subset reasoner can be configured utilizing the following parameters:

- *The ontology:* The ontology used for the reasoning process, this is typically the TBox.
- *Static knowledge base:* The static data that needs to be combined with the event data. There is an additional option to materialize this static data if this is not the case.
- *Data Streams:* The data streams that will produce event data.
- *Update Policies:* For each data stream, an update policy can be defined.
- *Queries:* Multiple queries can be registered, allowing the execute all the queries when new data arrives. There is also the option to execute specific queries.
- *Subset size:* The size of the subset can be set manually or can be automatically extracted from the registered ontology.

When data arrives, it is added to the registered streams and the assigned update policy will update the current view on that stream. The subset calculation will extract the necessary data and then the event data can be materialized. Once the materializations are combined, the knowledge base can be queried.

VII. EVALUATION

This section evaluates both the performance of the proposed approach and the completeness. The evaluation was conducted on a 16 core Intel Xeon E5520 @ 2.27GHz CPU with 12GB of RAM running on Ubuntu 16.04.

A. EVALUATION SET-UP

To evaluate the performance and completeness of the proposed approach, we evaluate our Subset approach in a real-life use case (as described in Section II) in Section VII-B and through an existing benchmark in Section VII-C. To evaluate the performance of the system, in Section VII-B1 and VII-C1 we compare with reasoners with the same expressivity and for purely illustrative purposes we compare in Section VII-B2 and VII-C2 with less expressive reasoners, however, they are unable to infer all answers.

The reasoners with same expressivity consist of the Pellet reasoner, the Stardog triple store[6], and two versions of the Hermit reasoner, one where we always materialize the whole knowledge base with Hermit and then query it (further referred to as 'hermit') and one where the basic graph patterns are evaluated inside the query engine with OWL 2 DL reasoning [37] (further referred to as 'owlbgp').

The selected reasoners with a lesser expressivity are RDFox, TrOWL and the jena incremental rule engine inferring the fragment of the ontology that can be represented as rules. The latter was incorporated because the C-SPARQL RSP engine [10] allows the use of any kind of rules within its query engine. We also incorporate the query rewriting method from StreamQR, i.e. we rewrote the queries for the $\mathcal{ELHIO}$ fragment of the ontology using the StreamQR rewriting techniques (further referred to as 'elhio'). We could not incorporate StreamQR directly, as it does not support static background data, such as the details regarding the staff members, hospital layout, etc.

We note that the results of these reasoners are not completely comparable as the expressivity of these approaches is lower than our Subset approach. These approaches could not produce all required results in the evaluated scenarios. In Section VII-B3 and VII-C3, we evaluate and discuss the correctness of the approaches.

B. USE-CASE EVALUATION

We implemented the scenario from the use case introduced in Section II. In each step of the scenario, an observation is sent and multiple queries are executed to determine the correct actions. To measure the scalability, we calculated how long the reasoning and querying took for each step in the scenario for a hospital ranging from 1 to 100 wards. We executed the scenario 35 times for each number of wards, dropped the first three and last 2 execution times and calculated the averages over the remaining 30 samples.

Table 1 shows the different amounts of axioms used for each number of wards and the complexity of the ontology.

[6]www.stardog.com

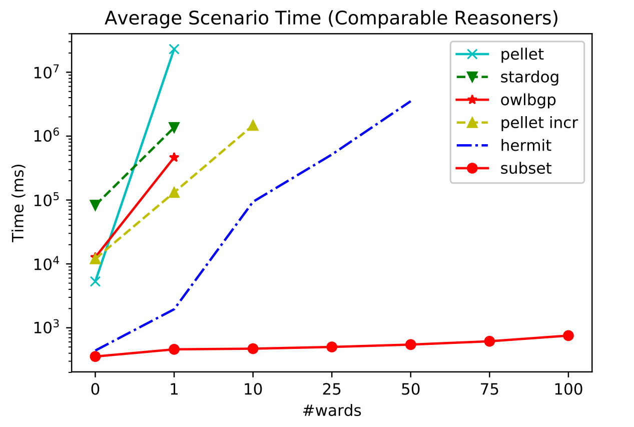

FIGURE 7. Comparison of the performance of various reasoners, with OWL2 DL expressivity, for increasing ABox data in the real-life use case.

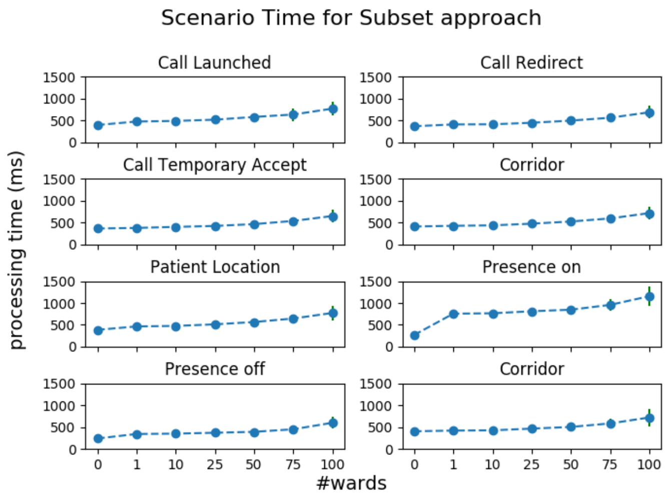

FIGURE 8. Performance of the Subset approach for increasing ABox data in the real-life use case.

With 0 wards we denote the case where we only consider the minimal set of individuals that allow to run the scenario, namely three locations and three persons (one patient and two staff members) each with their properties. This makes it the smallest ABox the scenario can run on. Note that the TBox depth of the ontology is three.

1) Performance Evaluation Comparable Reasoners

In Figure 7, we compare the execution time for the reasoners with OWL2 DL expressivity and our Subset approach. Note that the y-axis is in logarithmic scale. Furthermore, for the other approaches we use the same update policies used in the Subset approach, but update on the whole knowledge base.

It is clear that the other reasoners with the same expressivity do not scale very well in our scenario. As the number of wards and thus the ABox data increases, all of the approaches, except the Subset approach, become unusable slow for real-time decision making.

Since it is hard to see the performance of the Subset

TABLE 1. Summary of the ACCIO ontology for different number of wards.

#Wards:	0	1	10	25	50	75	100
Axioms	2169	2942	10643	19879	41438	63399	85094
Logical Axioms	1225	1835	7893	15151	32115	49396	66467
Individuals	88	250	1894	3871	8467	13147	17771
Classes	282						
Object Properties	131						
Data Properties	37						
DL Expressivity	SHOIQ(D)						

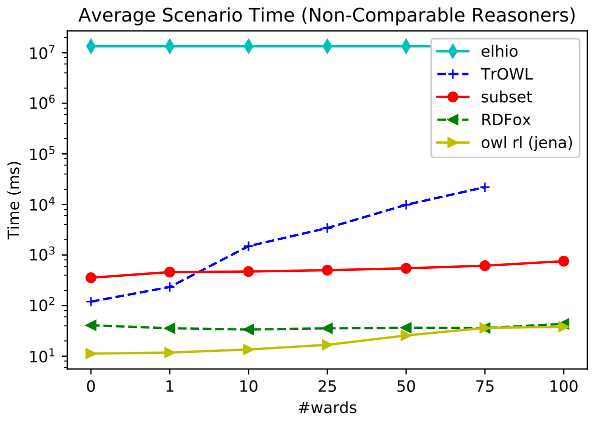

FIGURE 9. Comparison of the performance of various reasoners, with lower expressivity, for increasing ABox data in the real-life use case. Note that these lower expressive reasoner are not able execute the scenario correctly.

approach, we plot each step of the scenario for each number of wards in Figure 8. We can clearly see that the size of the ABox has a small influence on the execution time. This is because the subset can efficiently extract the necessary data. When the subset would be affected by an increase in data that could be logically separated, e.g. the number of wards, we can use a more fine-grained filtering, as described in Section VI-E, to separate the data in the subsets.

Note that in a preprocessing step, we need to materialize the static background knowledge for the Subset approach. This can be very time consuming for larger knowledge bases and is not taken into account in this comparison. However, since it only needs to be done once and the static knowledge typically does not change (often), it causes no real issues. In our use case, the static data contains the various locations in the hospital; the staff members and their capabilities; and the patients and their pathologies. Note that when the patients change too often, they can be modeled as a stream instead of static data.

2) Performance Evaluation Non-Comparable Reasoners

Figure 9 details the performance evaluation for the non-comparable reasoners. The less expressive reasoners (TrOWL, RDFox, owl rl (jena)) are more performant than the Subset approach. Even though TrOWL gets quickly caught up by the Subset approach when the ABox increases in size.

TABLE 2. Correctness of the different approaches. The correctness is calculated as the number of correct class assertions. The last column details whether each engine was able to correctly run the scenario.

Engine	Correctness	Scenario Correct
Subset	100%	yes
Rdfox	98.3%	no
Jena OWL-RL	97.2%	no
TrOWL	99.6%	no
ELHIO	87.1%	no

The OWL RL approaches are faster than the Subset approach, which can be expected as they only infer a fragment of the ontology. The fragment has been selected in such a way that it can be efficiently computed. An interesting observation is that rewriting of the queries based on an expressive ontology results in queries with many unions (between 430 and 138158, with an average of 34276 unions). The execution of these queries is very expensive due to the large number of unions. Even though our Subset approach is slower, yet more expressive, we see a similar trend in terms of scalability as for the OWL2 RL reasoners.

We point out to the reader that the most important comparison is to compare the Subset approach with the two hermit approaches (hermit and owlbgp) since they use the same reasoner. We chose to use Hermit within our subset reasoner since it is the most complete. However, the technique is reasoner independent. This means that the subset technique can be ported to other reasoners as well.

3) Correctness Evaluation

Since the Subset approach (without the extension from Section V-C) is an approximation technique, and we have included less expressive reasoners, we now discuss the correctness of each approach. To calculate the correctness we have run the scenario with the hermit reasoner and saved a materialized snapshot of the knowledge base in each step of the scenario. We view this as our baseline and compare for each approach the percentage of class assertions that have been correctly assigned to the individuals in the knowledge base.

Table 2 shows the correctness of each of the approaches. The last column indicates if the scenario was executed correctly. Our Subset approach is correct in this scenario, however none of the less expressive reasoners were able to execute the scenario correctly, even though the correctness level was already rather high. This is because the majority

TABLE 3. OWL2 RL coverage of the IoT labeled and the most prominent Smart Industry, Smart manufacturing, Pervasive Health and Social Media ontologies in the Linked Open Vocabularies repository (lov.linkeddata.es).

Ontology Name	OWL2 RL Coverage	#subclass def
The NASA Air Traffic Management Ontology	92.7%	263
Climate and Forecast (CF) standard names parameter vocabulary	95.6%	405
Data Value Vocabulary (DaVe)	50%	6
Ontology Modeling for Intelligent Domotic Environments	63.1%	2963
FIESTA-IoT Ontology	79.6%	598
Home Weather	49.2%	313
Iot-lite ontology	100%	11
The Machine-to-Machine Measurement (M3) Lite Ontology	82.2%	544
MobiVoc: Open Mobility Vocabulary	100%	17
SAN (Semantic Actuator Network)	64.3%	42
SAREF: the Smart Appliances REFerence ontology	76.2%	248
The SEAS Device ontology	58.3%	36
The SEAS Forecasting ontology	45.4%	11
The SEAS Time Ontology.	84.6%	13
Sensor, Observation, Sample, and Actuator (SOSA) Ontology	100%	0
Semantic Sensor Network Ontology	68.8%	80
Semantic Sensor Network Ontology (old version)	80.9%	89
VoCaLS: A Vocabulary and Catalog for Linked Streams	100%	13
Reference Architectural Model for Industry 4.0 (RAMI)	88.6%	35
Dem@Care Lab Ontology for Dementia Assessment	65.7%	213
The Event Ontology	69.7%	76
The CARESSES Ontology for Socially Assistive Robotics	53.5%	213

of concepts in the scenario do not always require expressive reasoning, however, to facility correct decision making in critical situation, e.g. correctly classify each call, expressive reasoning is necessary. Even though the frequency for the expressive reasoning might seem low, it is necessary to enable correct decision making as up to 40% of the scenario steps really depend on the results that require expressive reasoning (that cannot be inferred within the OWL2 RL fragment). The correctness results in Table 2 should thus be interpreted carefully as they indicate the correctness over the whole knowledge base. Even a small lack of correct results can lead to missing important events. This is clear for TrOWL, even with a correctness of 99.6% the scenario is not able to execute correctly.

In Table 3, we listed the OWL2 RL coverage for the IoT labeled and the most prominent Smart Industry, Smart Manufacturing, Pervasive Health and Social Media ontologies in the Linked Open Vocabularies repository (lov.linkeddata.es)[7]. In the right column, the table shows the number of subclass definitions that each ontology contains. Note that a class equivalence can be seen as two subclass definitions. The table shows that many of these ontologies consist definitions that are not fully covered by the OWL2 RL semantics. Even though the OWL2 RL reasoners are more performant, many existing ontologies still require higher expressive reasoning, such as OWL2 DL.

C. UOBM EVALUATION

The University Ontology Benchmark (UOBM) [38] is an existing reasoning benchmark consisting of universities of various sizes containing professors, assistant professors, undergraduate students, students, courses, publication, etc. The benchmark comes with four queries requiring expressive reasoning:

- **Q1:** Retrieve all women students. UOBM defines a woman college as: $WomanCollege \equiv School \wedge \forall hasStudent.Student \wedge \forall hasStudent.(\neg Man)$. Thus as a school where the students are not of the gender man. When a student is attending a woman college, the reasoner can infer that the student is female as *Man* and *Women* are defined as disjoint classes.
- **Q2:** Retrieve all people with many hobbies. UOBM defines people with many hobbies as: $PeopleWithManyHobbies \equiv \geq 3 \ \ like.\top$. They thus should like at least three things.
- **Q3:** Retrieve all people who love sports. UOBM defines people who like sports as: $SportsLover \equiv \exists like.Sports$. This means that there should exist a sport that the person likes.
- **Q4:** Retrieve all people with at least one hobby. People with a hobby are defined in UOBM as: $PeopleWithHobby \equiv person \wedge \geq 1 \ \ like.\top$. This means that the person should like at least one thing.

It is clear that the UOBM defines some complex concepts. However, compared to the ACCIO ontology used in Section VII-B, UOBM has a depth of one. UOBM is by default a static benchmark, so we extended the benchmark such that has a streaming characteristic[8]. This is done by allowing students to join a college over the year. This means that we extracted the generated students from the benchmark and stream them together with the courses they take, the college they attend, their interests and friends, etc. We have evaluated the average performance of processing the stream of student information over a university with 1, 2, 3, 4, 5 and 10 departments. Table 4 describes this in detail. As each student typically changes friends, courses and interests over the course of time, we have modeled each student and its updates as a distinct stream.

1) Performance Evaluation Comparable Reasoners

In Figure 10, we compare the average time to process a new student for the reasoners with OWL2 DL expressivity and our Subset approach. This indicates the reasoning time and the time to answer all four queries. Note that the y-axis is in logarithmic scale. It is clear that the other approaches do not scale well.

[7]We included all the ontologies that were accessible at the time of writing.

[8]The generator is available in the data folder on our Github page.

TABLE 4. Summary of the UOBM ontology for different number of university departments.

#Departments	**1**	**2**	**3**	**4**	**5**	**10**
Axioms	12405	26490	38849	52486	63949	133809
Logical Axioms	11943	25871	38075	51545	62870	131888
Individuals	1158	3141	4461	5937	7082	14063
Classes	113					
Object Properties	35					
Data Properties	9					
DL Expressivity	SHOIN(D)					

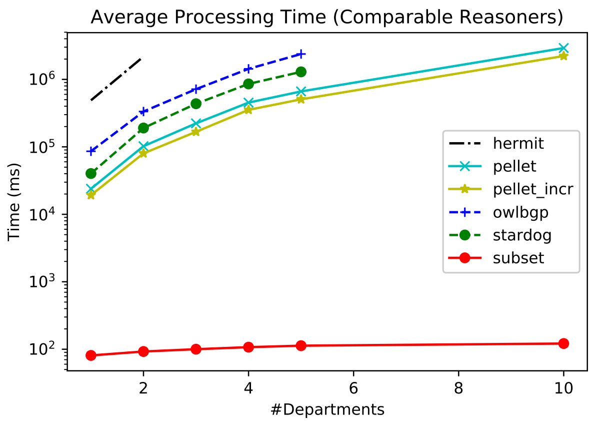

FIGURE 10. Comparison of the performance of various reasoners, with OWL2 DL expressivity, for increasing ABox data on the UOBM benchmark.

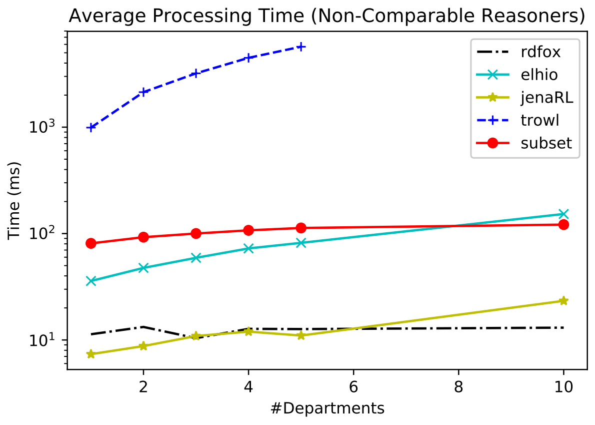

FIGURE 11. Comparison of the performance of various reasoners, with lower expressivity, for increasing ABox data on the UOBM benchmark.

2) Performance Evaluation non-Comparable Reasoners

In Figure 11 we show the comparison with reasoners with lower expressivity. It is clear that they are typically faster than our approach. However, as we will see in the completeness evaluation, in Section VII-C3, these approaches are fast in this scenario but are unable to infer a complete answer set. We even see that the rewriting approach, i.e. elhio, becomes slower as well due to the many unions in the rewritten queries. Furthermore, our Subset approach is faster than TrOWL.

3) Completeness Evaluation

Figure 12 shows the correctness of the approaches in the UOBM scenario in terms of the completeness of the given query answers. For each query, we evaluate the percentage of correctly derived results. It is clear that only HermiT, Pellet, OWLBGP and our Subset approach are able to provide correct answers. TrOWL is unable to provide correct answers to the first two queries, therefore it is not showing any results in the graph. Note that this holds for the other reasoners as well. If they could not answer the query at all, they are not shown in the graph as they produce 0% correct results. The other reasoners fail to make most of the derivations. Even though these reasoners are fast, they are incomplete in scenarios where expressive reasoning is required.

VIII. DISCUSSION

In this section, we discuss how our solutions tackles the set objectives, how Subset reasoning compares to the related work, how the evaluation results should be interpreted and the drawbacks and future work directions for Subset reasoning.

A. OBJECTIVES DISCUSSION

Looking back at the *Objectives* set in Section I-C, we can now discuss how our Subset reasoning approach tackles the various objectives:

1) *Heterogeneous data:* Our Subset reasoner can combine various heterogeneous sources by utilizing a common semantic model, i.e., an ontology.
2) *Event data:* The Subset approach can handle the addition and removal of event data through the use of its update policies that allows the system to have a clear view on the current context. Furthermore, the Subset approach utilizes an approximated extraction method to minimize the data to reason upon, decreasing reasoning time. Fast reasoning times are necessary in event-based system such that the system stays reactive and real-time decisions can be made.
3) *Large knowledge bases:* Many domains have large knowledge bases that need to be combined with the generated event data. However, reasoning over these large knowledge bases in combination with the changing event data might become slow. The Subset reasoner tackles this problem by exploiting the fact that the large knowledge bases typically consist of static data that does not change very often. Therefore, this data is materialized and due to the monotonicity property,

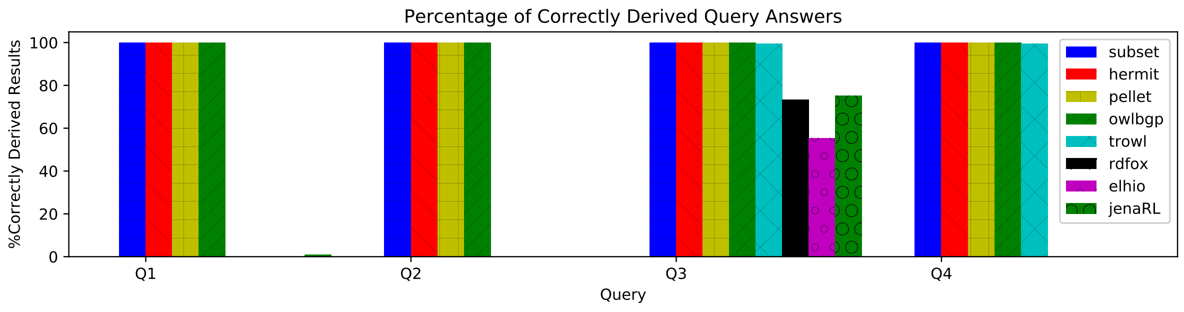

FIGURE 12. Correctness comparison UOBM evaluation for different reasoners.

TABLE 5. Comparisson of the related work to the Subset approach. (With W = windowing and UP = update policies.)

	Add/ Remove	Event Data	Expressive Reasoning	Large KB	View
Traditional Reasoners	x	/	x (OWL2 DL)	/	/
Pagoda	/	/	x (OWL2 DL)	x	/
RDFox	x	/	/ (OWL2 RL)	x	/
TrOWL	x	x	x (SHIQ)	/	/
RSP	x	x	/ (RDFS)	x	x (W)
StreamQR	x	x	/ (ELHIO)	/	x (W)
SubSet	**x**	**x**	**x** (OWL2 DL)	**x**	**x** (UP)

adding data will not lead to the removal of facts in the static data. The Subset reasoner extract data from the materialized knowledge base in order to compose a minimal subset to reason upon.

4) *Expressive reasoning :* Expressive reasoning is necessary to interpret many complex domains. The Subset approach supports OWL2 DL reasoning. It is able to perform this highly expressive reasoning over event data by computing a subset of data to reason upon.

B. RELATED WORK COMPARISON

Table 5 summarizes the related work and how they relate to our Subset approach. As we have seen in the evaluation in Section VII, traditional reasoners such as Pellet and Hermit are very expressive, however, they have problems processing event data in a timely fashion. They become very slow when the data increases and do not have any mechanisms to process event data besides adding and removing of data. The Pagoda reasoner tries to solve the performance problem by combining the Hermit reasoner with a less expressive OWL2 RL reasoner. However, Pagoda cannot be used with event data as it does not allow the addition and removal of data. RDFox is the fasted OWL2 RL reasoner currently available. It is very performant within its OWL2 RL fragment, however, as we have seen in Section VII-B3 and VII-C2, many use cases and ontologies require expressive OWL2 DL reasoning to correctly interpret the domain knowledge. Even though RDFox allows efficient addition and removal of data, it does not provide any mechanism such as windowing or update policies to process the event data and keep a view on the current context. TrOWL provides almost OWL2 DL expressivity, however, in Section VII-C2 we saw that even some unsupported constructions lead to incomplete answers. Furthermore, in the evaluation we have seen that TrOWL becomes rather slow when the background knowledge increases. RSP engines provide all the mechanisms to handle event data, however, due to their low expressivity, they cannot interpret complex domains. Query rewriting techniques such as provided by StreamQR can only inject a small part of their expressivity in the query and tend to become slow when the static data increases. Our Subset reasoner can perform expressive reasoning over event data, in combination with large knowledge bases by approximating a subset of data to reason upon. It is the only approach that fulfills all the set requirements.

C. EVALUATION DISCUSSION

In section VII-B1 we have shown that the Subset approach is a good candidate to make expressive reasoners more scalable and applicable in real-time scenarios were expressive reasoning is necessary. From section VII-B2 and VII-C2 it is clear that OWL2 RL reasoners, which are less expressive, are more efficient. However, as we have shown in Section VII-C3, these techniques are incomplete when expressive reasoning is required. Furthermore, in section VII-B3 we have shown that only a small fragment of the IoT labeled ontologies are completely covered by the OWL2 RL fragment. This implies that ontology designers either do not take efficiency into consideration when designing ontologies or ontology designers feel that the OWL2 fragments are lacking expressivity to model their domains.

We agree that some of the OWL2 DL definitions that cannot be represented in OWL2 RL, such as quantified number restrictions (e.g. there should be at exactly one person present in a certain room), could (in some form) be represented as standard existential quantifiers (e.g. there should be a person present in a certain room). However, such adaptations should be conducted very carefully as the semantics of the concepts are changing.

Thus, as expressive ontologies are still being designed and it is not trivial to convert expressive ontologies to their lesser expressive variants, techniques to efficiently reason

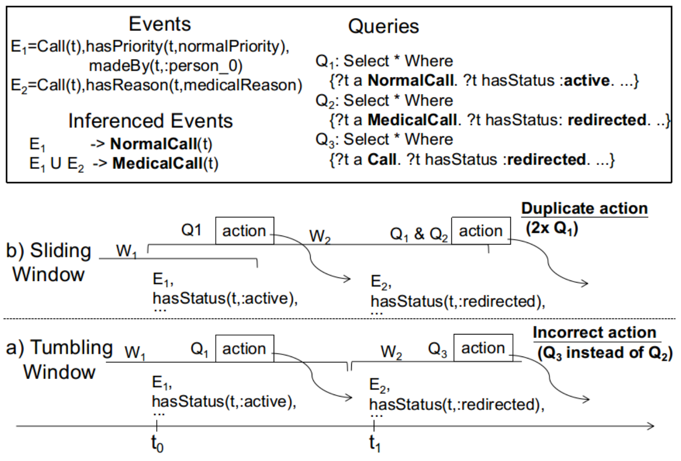
Events
E1=Call(t),hasPriority(t,normalPriority), madeBy(t,:person_0)
E2=Call(t),hasReason(t,medicalReason)
Inferenced Events
E1 -> NormalCall(t)
E1 U E2 -> MedicalCall(t)
Queries
Q1: Select * Where {?t a NormalCall. ?t hasStatus :active. ...}
Q2: Select * Where {?t a MedicalCall. ?t hasStatus: redirected. ..}
Q3: Select * Where {?t a Call. ?t hasStatus :redirected. ...}
b) Sliding Window
W1
Q1
action
W2
Q1 & Q2
action
Duplicate action (2x Q1)
E1, hasStatus(t,:active), ...
E2, hasStatus(t,:redirected), ...
a) Tumbling Window
W1
Q1
action
W2
Q3
action
Incorrect action (Q3 instead of Q2)
E1, hasStatus(t,:active), ...
E2, hasStatus(t,:redirected), ...
t0
t1

FIGURE 13. Complications of using windowing. The figure shows a typical dependency between events in the scenario. As reasoning is involved, the events need to be combined to infer the correct types that are used in the queries. However simply combining them, as in the sliding window, results in duplicate triggering queries (and thus duplicate actions). When the events are in different windows, incorrect actions are taken because the inferred information is missing.

upon expressive ontologies are still needed.

D. SUBSET REASONING LIMITATIONS & FUTURE WORK DIRECTIONS

A first limitation of the subset approach is that the data streams are handled separately, this means that if data from stream A has influence on data from stream B, that this will not be detected. This is because each stream is handled by its own update policy. This can be bypassed by combining various streams in the same update policy, however, this will have a negative impact on the performance. A second limitation is the fact that the update policies are currently not time based. This means that one has to explicitly remove facts in order to deprecate data. Time-based windows allow facts to be removed after a certain period of time, something which is not supported yet by our update policies. A third limitation is that our approach currently extracts ABox data only and takes the complete TBox each time into account, when performing reasoning.

In future work, we wish to further extend the subset approach and provide mechanisms to detect influences between streams. This would allow to efficiently process various streams that have influences on each other. Furthermore, we wish to further extend the update policies such that facts can be automatically removed after a certain period of time. This will eliminate the need to explicitly remove facts. This would also allow us to integrate aggregation mechanisms, as supported by RSP engines. We also wish to investigate mechanisms to efficiently minimize the TBox utilized in the reasoning process to further increase the reasoning performance.

E. PROBLEMS OF USING WINDOWING

In this section, we describe the implications of using windowing[9], instead of the update policies, has on the scenario, as visualized in Figure 13. Say we have two events (event E_1 and event E_2), E_1 can be inferred through reasoning as a *NormallCall*, when adding the information described in E_2, that says that the call has a medical reason, to E_1, the call can now be inferred as a *MedicalCall*. These kinds of dependencies occur throughout the use case. Different queries are executed in a certain order on these events and when one of the queries triggers, a certain action is executed. When E_1 arrives, describing a new call has been launched by a certain patient, query Q_1 that selects staff members for new calls (status active) is executed and notifies the selected staff member. This is step 1 (Call Launched) in the scenario. In step 2 (Call Redirect) the staff member is busy and redirects the call and indicates that the call has a medical reason. This is described in E_2. Normally, query Q_2 should select a new staff member for medical calls that have been redirected.

However, when using windowing, the dependencies between the events can be lost. Figure 13 shows both the problems for both tumbling and sliding windows. When using a tumbling window (Figure 13 a), Q_1 is executed upon E_1 and as a result E_2 is sent. However, in the next window W_2, the information regarding the call described in E_1 is lost and now it is unknown that the call was a *NormalCall* and therefore Q_2 will not be executed. Query Q_3 that redirect any kind of call will match and an incorrect action will be taken. When using a sliding window (Figure 13 b), both events can be comprised in the same window, however, as the call keeps its initial status (active) query Q_1 will trigger for a second time and duplicate actions will be taken. This problem occurs due to the lack of an update policy that should overwrite the call status.

IX. CONCLUSION AND FUTURE WORK

In this paper, we presented a technique that allows to bridge the gap between volatile data and expressive reasoning. Our technique maintains a materialized view on the ontology ABox and uses a subset approximation to efficiently update the materialized view. To define how these updates should happen, we introduced the notion of update policies. The subsetting enables a scalable system even with highly increasing ABoxes.

In our future work, we will investigate the possibility to perform aggregations, as supported by RSP, within the update policies. Furthermore, we will investigate the integration of module extraction techniques [21] to, besides minimizing the ABox, minimize the used TBox within the reasoning process. Our technique achieves a speed-up of up 10 times for small ABoxes and more than 1000 for larger ones.

We show that the subsetting technique is a valid tool to enable expressive reasoning in time-critical scenarios, allow-

[9]There exist various entailment regimes [1], we explain graph-level entailment as it is currently the most commonly used.

ing real-time systems to make complex decisions based on expressive reasoning solutions.

Acknowledgment This research was funded by the IWT O&O ORCA and Televic Healthcare.

REFERENCES

[1] D. Dell'Aglio, E. Della Valle, F. van Harmelen, and A. Bernstein, "Stream reasoning: A survey and outlook," Data Science, no. Preprint, pp. 1–24, 2017.

[2] A. Margara, J. Urbani, F. Van Harmelen, and H. Bal, "Streaming the web: Reasoning over dynamic data," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 25, pp. 24–44, 2014.

[3] M. I. Ali, N. Ono, M. Kaysar, Z. U. Shamszaman, T.-L. Pham, F. Gao, K. Griffin, and A. Mileo, "Real-time data analytics and event detection for iot-enabled communication systems," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 42, pp. 19–37, 2017.

[4] F. Ongenae, P. Duysburgh, N. Sulmon, M. Verstraete, L. Bleumers, S. De Zutter, S. Verstichel, A. Ackaert, A. Jacobs, and F. De Turck, "An ontology co-design method for the co-creation of a continuous care ontology," Applied Ontology, vol. 9, no. 1, pp. 27–64, 2014.

[5] S. Bergamaschi, S. Castano, M. Vincini, and D. Beneventano, "Semantic Integration of Heterogeneous Information Sources Using a Knowledge-Based System," Data & Knowledge Engineering, vol. 36, pp. 215–249, 2001.

[6] T. Strang and C. Linnhoff-Popien, "A context modeling survey," in Workshop on advanced context modelling, reasoning and management, UbiComp, vol. 4, 2004, pp. 34–41.

[7] U. Hustadt, B. Motik, and U. Sattler, "Data complexity of reasoning in very expressive description logics," in IJCAI, vol. 5, 2005, pp. 466–471.

[8] L. Al-Jadir, C. Parent, and S. Spaccapietra, "Reasoning with large ontologies stored in relational databases: The OntoMinD approach," Data & Knowledge Engineering, vol. 69, no. 11, pp. 1158–1180, 2010.

[9] A. Hogan, A. Harth, and A. Polleres, "Saor: Authoritative reasoning for the web," The Semantic Web, pp. 76–90, 2008.

[10] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus, "Querying RDF streams with C-SPARQL," SIGMOD Record, 2010.

[11] D. Le-Phuoc, M. Dao-Tran, J. Xavier Parreira, and M. Hauswirth, A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 370–388.

[12] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic, "EP-SPARQL: a unified language for event processing and stream reasoning," 2011, pp. 635–644.

[13] R. Shearer, B. Motik, and I. Horrocks, "HermiT: A Highly-Efficient OWL Reasoner." in OWLED, vol. 432, 2008, p. 91.

[14] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, and J. Banerjee, "Rdfox: A highly-scalable RDF store," in ISWC 2015 , Proceedings, Part II, 2015, pp. 3–20.

[15] H. Stuckenschmidt, S. Ceri, E. Della Valle, and F. Van Harmelen, "Towards expressive stream reasoning," in Semantic Challenges in Sensor Networks, 24.01. - 29.01.2010, 2010.

[16] E. Della Valle, S. Schlobach, M. Krötzsch, A. Bozzon, S. Ceri, and I. Horrocks, "Order matters! harnessing a world of orderings for reasoning over massive data," Semantic Web, vol. 4, no. 2, pp. 219–231, 2013.

[17] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," Web Semantics: science, services and agents on the World Wide Web, vol. 5, no. 2, pp. 51–53, 2007.

[18] Y. Zhou, B. Cuenca Grau, Y. Nenov, M. Kaminski, and I. Horrocks, "Pagoda: Pay-as-you-go ontology query answering using a datalog reasoner," Journal of Artificial Intelligence Research, vol. 54, pp. 309–367, 2015.

[19] E. Thomas, J. Z. Pan, and Y. Ren, "TrOWL: Tractable OWL 2 reasoning infrastructure," in Extended Semantic Web Conference. Springer, 2010, pp. 431–435.

[20] J.-P. Calbimonte, J. Mora, and O. Corcho, "Query rewriting in rdf stream processing," in International Semantic Web Conference. Springer, 2016, pp. 486–502.

[21] A. A. Romero, M. Kaminski, B. C. Grau, and I. Horrocks, "Module extraction in expressive ontology languages via datalog reasoning," J. Artif. Int. Res., vol. 55, no. 1, pp. 499–564, Jan. 2016.

[22] A. Schlicht and H. Stuckenschmidt, "Criteria-based partitioning of large ontologies," in Proceedings of the 4th international conference on Knowledge Capture. ACM, 2007, pp. 171–172.

[23] C. Anagnostopoulos and S. Hadjiefthymiades, "Enhancing situation-aware systems through imprecise reasoning," IEEE Transactions on Mobile Computing, vol. 7, no. 10, pp. 1153–1168, 2008.

[24] M. Hassanalieragh, A. Page, T. Soyata, G. Sharma, M. Aktas, G. Mateos, B. Kantarci, and S. Andreescu, "Health monitoring and management using Internet-of-Things (IoT) sensing with cloud-based processing: Opportunities and challenges," in Services Computing (SCC), 2015 IEEE International Conference on. IEEE, 2015, pp. 285–292.

[25] D. Macagnano, G. Destino, and G. Abreu, "Indoor positioning: A key enabling technology for IoT applications," in Internet of Things (WF-IoT), 2014 IEEE World Forum on. IEEE, 2014, pp. 117–118.

[26] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, "Owl 2 web ontology language primer," W3C recommendation, vol. 27, no. 1, p. 123, 2009.

[27] F. Baader, D. Calvanese, D. McGuinness, P. Patel-Schneider, and D. Nardi, The description logic handbook: Theory, implementation and applications. Cambridge university press, 2003.

[28] R. Kontchakov and M. Zakharyaschev, "An introduction to description logics and query rewriting," in Reasoning Web International Summer School. Springer, 2014, pp. 195–244.

[29] I. Horrocks, O. Kutz, and U. Sattler, "The even more irresistible sroiq." Kr, vol. 6, pp. 57–67, 2006.

[30] I. Horrocks, U. Sattler, and S. Tobies, "Practical reasoning for expressive description logics," in International Conference on Logic for Programming Artificial Intelligence and Reasoning. Springer, 1999, pp. 161–180.

[31] "A tableau decision procedure for SHOIQ, author=Horrocks, Ian and Sattler, Ulrike, journal=Journal of automated reasoning, volume=39, number=3, pages=249–276, year=2007, publisher=Springer."

[32] D. Tsarkov and I. Horrocks, "Fact++ description logic reasoner: System description," in International Joint Conference on Automated Reasoning. Springer, 2006, pp. 292–297.

[33] V. Haarslev, K. Hidde, R. Möller, and M. Wessel, "The racerpro knowledge representation and reasoning system," Semantic Web, vol. 3, no. 3, pp. 267–277, 2012.

[34] E. A. Emerson, "Temporal and modal logic." Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), vol. 995, no. 1072, p. 5, 1990.

[35] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus, "Incremental reasoning on streams and rich background knowledge," in Extended Semantic Web Conference. Springer, 2010, pp. 1–15.

[36] B. Motik, Y. Nenov, R. E. F. Piro, and I. Horrocks, "Incremental update of datalog materialisation: the backward/forward algorithm." in AAAI, 2015, pp. 1560–1568.

[37] I. Kollia, B. Glimm, and I. Horrocks, "Sparql query answering over owl ontologies," in Extended Semantic Web Conference. Springer, 2011, pp. 382–396.

[38] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu, "Towards a complete owl ontology benchmark," in European Semantic Web Conference. Springer, 2006, pp. 125–139.