Evaluation and Optimized Usage of OWL 2 Reasoners in an Event-based eHealth Context.

Pieter Bonte[1], Femke Ongenae[1], Jeroen Schaballie[1], Ben De Meester[1], Dörthe Arndt[1], Wim Dereuddre[2], Jabran Bhatti[2], Stijn Verstichel[1], Ruben Verborgh[1], Rik Van de Walle[1], Erik Mannens[1], and Filip De Turck[1]

[1] Ghent University - iMinds, Gaston Crommenlaan 8, 9000 Ghent, Belgium
Pieter.Bonte@intec.ugent.be
[2] Televic HealthCare, Leo Bekaertlaan 1, 8870 Izegem

Abstract This paper evaluates the performance of the OWL 2 reasoners Pellet and HermiT in an eHealth context where most of the ABox is considered static and discrete transient events describing the environment are incrementally added and processed. The considered use case is the assignment of tasks and calls to nurses. To provide personalized and optimized care, the selection process utilizes reasoning to make intelligent assignment decisions based on the available information. This has been implemented using multiple SPARQL-queries to enable easy adaptation of the assignment algorithm. Since limited time is available to perform the assignments, the decision should be made in at most five seconds. An analysis of the performance and scalability of the reasoners is presented. To deal with the limited time frame, several optimizations are suggested, which exploit that most of the ABox is considered static.
Keywords: eHealth, Evaluation, Event-based, OWL2 Reasoners

1 Introduction

To provide personalized care for patients, task and call assignment systems benefit from considering as much information as possible. Most of this info can be considered relatively static, e.g., the patient's profile and pathology and the competences of the staff. Discrete events representing tasks, calls and changes to the environment, e.g., person location updates, are incrementally added and processed. Scalable reasoning on this static data, which is incrementally updated with event data, is thus required to assign the most appropriate staff to tasks and calls. The selection procedure, represented as a decision tree, has been implemented using multiple SPARQL-queries, allowing easy adaptation of the algorithm. Since the selection procedure should happen as fast as possible, the allowed decision time has been limited to five seconds by domain experts.

In this paper, the developed task and call assignment reasoning platform is evaluated in terms of scalability using the OWL-2 reasoners Pellet [10] and HermiT [8] based on hospital data with an increasing number of wards. Moreover, several optimizations to speed up the reasoning are proposed and evaluated. These optimizations are able to deal with the event-based and time-constraint scenarios and are able to execute the numerous SPARQL-queries efficiently.

2 The Scenario

The considered scenario consists of a hospital with wards, containing patients and care staff. The patients execute calls to receive medical aid. The locations of the medical staff are automatically tracked. When a patient is in need of aid, the decision tree is checked to determine the most suited staff member, based on the patient's pathology and profile, the location of the care staff, etc. The scenario consists of following steps, with the performed reasoning actions between brackets:
Call Launched: A patient launches a call (select nurse)
Call Redirect: The nurse indicates that she is busy (select new nurse)
Call Temporary Accept: The new nurse accepts the call (update call status)
Corridor: The nurse moves towards the room of the patient (update location)
Patient loc: The nurse arrives in the room (update location & turn on lights)
Presence On: The nurse logs into the terminal (update call status & turn on lights)
Presence Off: The nurse logs out (update call status & turn off lights)
Corridor loc: The nurse leaves the room (update location & turn off lights)

In the considered eHealth use case, the static data consists of the hospital configuration and profile information of the patients and care staff. The dynamic data are typically calls by the patients to receive aid, updates of the status of the call or location updates by the staff.

3 The Accio Ontology

To represent the eHealth knowledge, the ACCIO ontology[3] is used. An elaborate description can be found in Ongenae, et al. [7].

We evaluate the scalability of the OWL 2 reasoners by executing the scenario over an increasing number of wards. The details of the ontology loaded with the preliminary data for each number of wards is summarized in Table 1. The loaded data consists of the configuration of the different wards, including the personnel and patient information. Note that these are the numbers of the static part of the ABox. During the scenario, the TBox is considered static.

#Wards:	**1**	**10**	**25**	**50**	**75**	**100**
Axioms	3412	11113	20349	41098	63869	85564
Logical Axioms	2109	8167	15426	32389	49670	66741
Individuals	270	1913	3890	8464	13166	17790
Classes	332					
Object Properties	182					
Data Properties	51					
DL Expressivity	SHOIQ(D)					

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

[3] http://users.intec.ugent.be/pieter.bonte/ontology/accio.html

4 Implementation

To implement the scenario, we utilized the ModulAr, Service, Semantic & Flexible Platform (MASSIF), a data-driven platform that allows the easy development and collaboration of (ontology-based) services. Each service performs a distinguished reasoning task. Services exchange their knowledge over a Semantic Communication Bus (SCB) [2]. A detailed description of the MASSIF platform can be found in De Backere, et al. [1].

Four Services, performing different reasoning tasks, were implemented. These are elaborated below, in followed by the number of queries needed to implement their logic.
Presence Service: tracks the location of the staff – *#queries: 2.*
Status Call Service: tracks the status of the calls – *#queries: 4.*
Light Service: handles the lights in the patient rooms – *#queries: 9.*
Help Selection Service: handles the staff assignment – *#queries: 200.*

Even though the *Help Selection Service* implements 200 queries, the real number of executed queries depends on the number of branches that need to be checked in the decision tree, which represents the nurse assignment algorithm.

The OWL API [4] is used to internally represent the ontology in the MASSIF platform. It provides an *OWLReasoner*-interface, offering an uniform access point to various reasoners. In this paper we evaluate the most popular reasoners implementing the interface. Unfortunately, the OWL API does not provide any SPARQL support. To resolve this matter a Jena model [5] was used that allows SPARQL queries through Jena ARQ. This model requires synchronization with the OWL API. Only Pellet is able to convert its internal knowledge base to Jena. Utilizing HermiT or a more optimized use of Pellet requires a direct conversion from OWL API to Jena, this is achieved by writing the ontology to a outputstream in RDF/XML format and reading this stream in Jena.

Different approaches were evaluated for using the reasoners, of which the distinctive steps are visualized in Figure 1. Two flows can be discerned. A flow at the top describing the precomputation steps at start-up and one at the bottom describing the reasoning steps during the execution of the scenario. Before discussing them in detail, we elaborate the on two implemented optimizations for executing numerous queries in an event-based context.

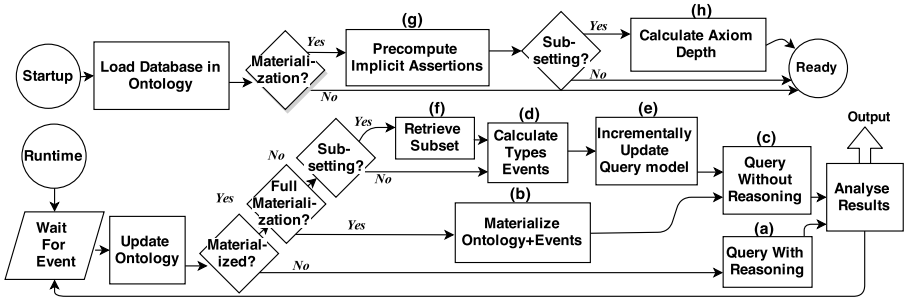


Figure 1. Workflow depicting the various approaches

4.1 Materialization

Materialization is the process of precomputing key sets of implicit assertions in the knowledge base and is frequently employed by semantic query and reasoning engines to improve query performance [6]. Doing so requires more storage, but it allows easy look-up at run-time [9], because it can bypass reasoning when executing queries. This step is depicted in Figure 1 as (g).

Adding additional event data (ABox) to a materialized ontology requires realization, i.e., computing the direct types of the added individuals. Based on a previous classification, the class hierarchy can be used to retrieve all types.

4.2 Subset Reasoning

When adding new individuals to a materialized ontology, subset reasoning retrieves a subset of ABox data in such a way that all necessary data to calculate the types of the new individuals can be achieved with a minimal data set. The size of the subset is ontology dependent and determined through a precomputation step, which finds the TBox axiom with the largest depth. This is depicted in Figure 1 as step (h). The depth of an axiom is similar to the modal depth and defines the deepest nesting of the operators. A formal definition can be found below. We define the axiom depth as d, R as the roles and C as the concepts.

$$\begin{aligned} \theta &= C|\forall R.\theta|\exists R.\theta & d(C) &= 0 \\ d(\theta_1 \wedge \theta_2) &= max(d(\theta_1), d(\theta_2)) & d(\forall R.\theta) &= 1 + d(\theta) \\ d(\theta_1 \vee \theta_2) &= max(d(\theta_1), d(\theta_2)) & d(\exists R.\theta) &= 1 + d(\theta) \end{aligned} \tag{1}$$

The calculated depth defines the size of the dataset necessary to calculate the types of new individuals in the materialized ontology. If we view the ontology as a graph with the individuals as vertexes and the relations as edges, the subset of data needed to calculate the types is a subtree of the graph with as root the individual and as depth the calculated axiom depth. If the types of multiple individuals need to be calculated, a union of subtrees can be considered.

A subset is sufficient to determine the types of an individual. Only the individuals/literals with whom the given individual has a relation have influence, considering the calculated depth and the fact that all needed inferred data has been calculated in a previous materialization step. For this approach to preserve completeness, the TBox describing the new individuals should be part of an ontology definition T that can be seen as an extension of the static ontology definition T' and if T is local, it does not yield new consequences in T'. A definition of locality can be found in Grau, et al. [3]. Transitive relations to leafs in the subtree could also cause incompleteness, since the transitivity could possibly not be fulfilled because of the limited data in the subset. However, this never occurs in our scenario. The maximum axiom depth for the ACCIO ontology is three.

Compared to existing modularization techniques [11], our technique only extracts ABox data, preserving the TBox allows dynamic and performant extraction at runtime. Since the ontology has been materialized, the extracted data can be limited to the data that directly influences the calculation of the new types. All other data has been inferred in a previous step.

4.3 The Approaches

The different approaches used to perform the reasoning are discussed below. Each approach is depicted in Figure 1 as a sequence of steps.

1. Pellet is used to perform the necessary reasoning each time a query is executed. Pellet performs the conversion of its internal knowledge base to a Jena Model each time an event arrives, but is able to cache intermediate results. This is depicted in Figure 1 as step (a).

2. To eliminate the reasoning at query time, a full materialization of the ontology is calculated at arrival of an event and translated from the OWL API to the Jena Model. Jena ARQ allows querying without any reasoning. This is depicted as step (b) and has been evaluated with the HermiT reasoner.

3. Instead of calculating the whole inferred model each time an event arrives, the static data is materialized as a pre-computation step. For each individual in the arriving data, its types are calculated based on the materialized ontology and the OWL API and the Jena model are incremented with this inferred knowledge. This is shown as steps (d)-(e)-(c) and has been evaluated with Pellet and HermiT.

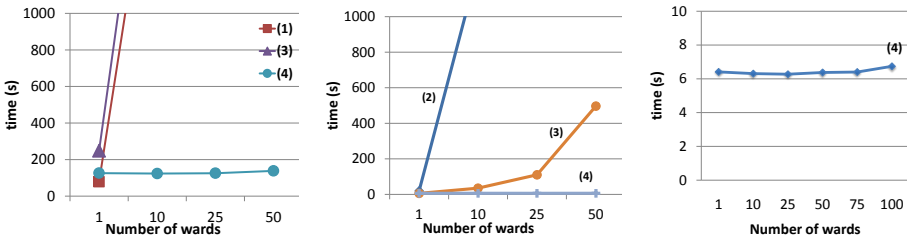
4. To determine the types of the arriving events, it is not necessary to analyze the whole ontology. Starting from a materialized ontology, we compute the subset of data that has influence on the calculation of the types of the newly arrived individuals, by utilizing the subset reasoning explained in Section 4.2. This step is depicted as steps (f)-(d)-(e)-(c) and has been evaluated with Pellet and HermiT.

It is important to note that in approaches 3 and 4, the completeness of the materialized ontology might become partly lost when the new events yield consequences in the static data. This is because the calculation of the types of the event data does not lead to a recalculation of the materialized static data. However, this is never the case in the eHealth scenario discussed in this paper.

5 Evaluation Set-up and Results

The scalability of the reasoners is evaluated by increasing the number of wards, resulting in a growing ABox. Each ward consists of 10 rooms, two patients asking for aid and three nurses. Each approach and each number of wards was evaluated 35 times. The first three and last two results were dropped to eliminated the influence of the warm-up and cooling down period. The evaluation was done on a Debian server with an Intel Xeon CPU E5620 2.40GHz with 12 GB of memory.

Figure 2 summarizes the performance of the reasoners. The corresponding approach is indicated between brackets. On the Y-axis the time to complete the whole scenario, which contains multiple reasoning steps, is indicated. Evaluating over the sum of the various scenario steps, allows us to gain a clear understanding of the different trends for the various approaches. Since the results for Pellet do not meet the time constraint of five seconds for 1 ward, more than 10 wards were not evaluated. As for HermiT, materializing the whole ontology every time an event arrives is not feasible. Since the execution time for ten wards did not come close to the time constraints, the evaluation was not continued. Approach 3 is



(a) Pellet Reasoner (b) HermiT Reasoner (c) HermiT + Subset

Figure 2. Evaluation of the various approaches

more performant than the Pellet approaches but still scales not well because it takes the whole dataset into consideration when reasoning.

The subsetting approach is the fastest and meets the time constraints. Therefor it is shown in detail in Figure 2 (c). It is clear that the size of the dataset has limited influence on the reasoning times due to the extraction of the minimal dataset through subsetting. Partly losing the completeness of the ontology has huge performance benefits. We therefore analyze these results more in depth in Figure 3 which visualizes the average times for each service in each scenario step for a fixed number of wards (here 75). The presented times are the total service time. For the *HelpSelectionService* 55% of the time is spent on reasoning, 42% on quering and less than 3% on conversion. The execution time for the various scenario steps differ. This is due to the fact that not all steps require the same amount of reasoning or queries. We can see that the *HelpSelectionService* takes the longest in the first two steps, as this service checks the biggest decision tree.

6 Conclusion and Future Work

In this paper the performance and scalability of the HermiT and Pellet reasoner in an event-based eHealth scenario were evaluated. Scalable reasoning over relatively static data that is incrementally updated with event data is reached by limiting the data the reasoners need to calculate the types of the event data. It is clear that partly losing the completeness of the ontology has huge performance benefits. The technique can be exploited in cases where performance is critical and where the dynamic part of the ontology which describes the events has no or limited consequences on the rest of the ontology. Furthermore, it was shown that HermiT is more performant in calculating the inferred types than the Pellet reasoner. In future work, we will focus on adapting the subset-algorithm to reclaim completeness of the whole ontology by incrementally increasing the subset if possible changes to the static data have been detected.

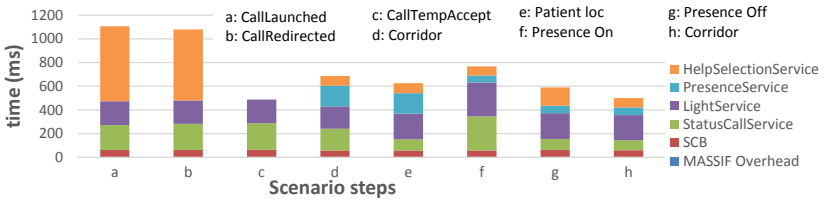


Figure 3. Evaluation of the HermiT reasoner for the subset approach

References

1. De Backere, F., Ongenae, F., Van den Abeele, F., Nelis, J., Bonte, P., Clement, E., Philpott, M., Hoebeke, J., Verstichel, S., Ackaert, A., et al.: Towards a social and context-aware multi-sensor fall detection and risk assessment platform. Computers in biology and medicine (2014)
2. Famaey, J., et al: An ontology-driven semantic bus for autonomic communication elements. In: Brennan, R., Fleck, J., van der Meer, S. (eds.) Lecture Notes in Comput. Sci. vol. 6473, pp. 37–50. Springer Verlag Berlin (2010)
3. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: A logical framework for modularity of ontologies. In: IJCAI. vol. 2007, pp. 298–303 (2007)
4. Horridge, M., Bechhofer, S.: The owl api: A java api for owl ontologies. Semantic Web 2(1), 11–21 (2011)
5. McBride, B.: Jena: Implementing the rdf model and syntax specification. In: SemWeb (2001)
6. Narayanan S, Catalyurek U, K.T.S.J.: Parallel materialization of large aboxes. In: Symposium on Applied Computing. vol. 2009, pp. 1257–1261
7. Ongenae, F., Bleumers, L., Sulmon, N., Verstraete, M., Van Gils, M., Jacobs, A., De Zutter, S., Verhoeve, P., Ackaert, A., De Turck, F.: Participatory design of a continuous care ontology (2011)
8. OXFORD, U.O.: Hermit reasoner (2014), http://hermit-reasoner.com
9. Rabbi, F., MacCaull, W., Faruqui, R.U.: A scalable ontology reasoner via incremental materialization. Proceedings of CBMS 2013 - 26th IEEE International Symposium on Computer-Based Medical Systems pp. 221–226 (2013)
10. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Web Semantics: Science, Services and Agents on the World Wide Web 5(2), 51–53 (2007), http://www.sciencedirect.com/science/article/pii/S1570826807000169
11. Tsarkov, D.: Improved algorithms for module extraction and atomic decomposition. In: 25th International Workshop on Description Logics. p. 345. Citeseer (2012)