Ontology Reasoning using Rules in an eHealth Context

Dörthe Arndt[1], Ben De Meester[1], Pieter Bonte[2], Jeroen Schaballie[2], Jabran Bhatti[3], Wim Dereuddre[3], Ruben Verborgh[1], Femke Ongenae[2], Filip De Turck[2], Rik Van de Walle[1], and Erik Mannens[1]

[1] Ghent University – iMinds – Multimedia Lab, Belgium
{doerthe.arndt, ben.demeester}@ugent.be
[2] IBCN research group, INTEC, Ghent University – iMinds, Belgium
[3] Televic Healthcare, Belgium

Abstract. Traditionally, nurse call systems in hospitals are rather simple: patients have a button next to their bed to call a nurse. Which specific nurse is called cannot be controlled, as there is no extra information available. This is different for solutions based on semantic knowledge: if the state of care givers (busy or free), their current position, and for example their skills are known, a system can always choose the best suitable nurse for a call. In this paper we describe such a semantic nurse call system implemented using the EYE reasoner and Notation3 rules. The system is able to perform OWL-RL reasoning. Additionally, we use rules to implement complex decision trees. We compare our solution to an implementation using OWL-DL, the Pellet reasoner, and SPARQL queries. We show that our purely rule-based approach gives promising results. Further improvements will lead to a mature product which will significantly change the organization of modern hospitals.

Keywords: Notation3, eHealth, OWL 2 RL

1 Business Case

Our business case is a nurse call system in a hospital. The system is aware of certain details about personnel and patients. Such information can include: personal skills of a staff member, staff competences, patient information, special patient needs, and/or the personal relationship between staff members and patients. Furthermore, there is dynamic information available, as for example the current location of staff members and their status (busy or free). When a call is made, the nurse call system should be able to assign the best staff member to answer that call. The definition of this "best" person varies between hospitals and can be quite complex. Our system should thus be easily adjustable, but also very fast in taking a decision. The system additionally controls different devices. If for example staff members enter a room with a patient, a decent light should be switched on; if they log into the room's terminal, they should have access to the medical lockers in the room. Especially hospitals are interested in that kind of system as it enables them to organize their work in a more efficient way:

- Busy nurses get distracted less. They only receive a call if everyone else is also occupied or if the new task is more important than the task they are currently performing.
- The system allows giving preference to staff members who are close to the caller. This prevents nurses from covering unnecessary big distances in their anyhow stressful and physically exhausting daily work.
- If the system is aware of the reason for a call, it can immediately assign nurses with the required skills. Thus, no time is lost by first calling other staff members who then would have to ask for additional help.
- The system can prefer staff members who already know the patient and have a trust relationship with him. This increases the satisfaction of the patient. At the same time, it also saves time for the caregiver, who is in such cases already familiar with the patient's needs and condition.
- The system is universal, i.e., electronic devices in the hospital can be controlled as well.
- The system is adaptable, i.e., hospitals can add their own requirements and priorities.

2 Technological Challenges

An event-driven system as described above has to fulfill certain requirements. The system should:

Scalability cope with data sets ranging from 1000 to 100 000 relevant triple (i.e., triples necessary to be included for the reasoning to be correct);
Semantics be able to draw conclusions based on the information it is aware of;
Functional complexity implement deterministic decision trees with varying complexities;
Configuration have the ability to change these decision trees at configuration time; and
Real-time return a response within 5 seconds to any given event.

There are several options to implement a nurse call system as described above. Following a more classical approach, the system could be written in an object-oriented programming language such as Java or C++. An implementation like this can easily fulfill the real-time and scalability constraints. But such systems are traditionally hard-coded: they are implemented for a specific use case, and even though they might be able to support the required functional complexity, this implementation would be static. The possibility to configure complex decision trees as postulated by the complexity requirement is rather hard to fulfill using traditional programming. Even more difficult to satisfy is the semantic requirement: most object oriented languages do not support enough logic to "understand" the available information. Knowledge must be stated explicitly, as even simple connections between statements such as "*nurse x has location y*" and "*y is location of nurse x*" cannot be found easily.

Especially the last argument motivates us to solve the described problem using semantic web technologies as they natively fulfill the semantics requirement. Knowledge can be represented in an OWL ontology which is understood by OWL-DL reasoners such as for example Pellet [6]. Complex decision trees can be handled by subsequent SPARQL queries. It is easy to add new queries or to change the order of existing queries and to thereby accommodate for the configuration constraint. But our tests have shown that such systems inherently are not fast and reliable enough to fulfill the scalability and real-time requirements. For bigger amounts of data or too complex decision trees, the reasoning times of traditional OWL-DL reasoners grow exponentially, which is not scalable.

To keep the benefits of an OWL-DL based implementation in a scalable and real-time way, we propose a rule-based solution. By using OWL 2 RL rules and resolution instead of classical tableaux reasoning, we can significantly decrease reasoning times and still cover a major subset of OWL 2. This approach benefits from the high performance of rule based reasoners. Even for bigger datasets the reasoning times are still faster than with the OWL-DL based approach. Also, complex decision trees can directly be implemented in rules. As rules are the most natural representations of such trees, it is easy for a user to understand and change certain priorities or to configure new rules. A further advantage of our approach is that all logic is represented in one single way. Instead of OWL plus SPARQL, we can implement the business case by only employing Notation3 Logic (N3). With the aforementioned system, we can meet all necessary requirements.

3 Rule-based Solution

In this section, we further explain our solution in three parts. First, we focus on the technical background and the technologies used. In the second part we describe how we could improve reasoning times for our use case by employing OWL 2 RL rules written in N3. In the third part we show example rules from our decision trees and discuss options to change these rules.

3.1 Background

Our solution improves on a more classical implementation which employs the OWL 2 reasoner Pellet [6], and where the decision tree was implemented via SPARQL queries. All knowledge was represented using the ACCIO ontology, which is described by Ongenae et al. [5]. Knowledge could either be fixed or dynamic, and updated via an event. Fixed knowledge is, for example, information about skills of staff members, or patients' preferences. Dynamic knowledge is, for example, the movement of a nurse to a certain location, or a new call which is made by a patient.

Our new implementation keeps the knowledge representation of the first attempt but replaces SPARQL and OWL by Notation3 Logic (N3) [1]. This logic forms a superset of RDF and extends the RDF data model by formulas (graphs),

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

{?C rdfs:subClassOf ?D. ?X a ?C} => {?X a ?D}.
```

Listing 1: OWL-RL rule for `rdfs:subClassOf` class axiom in N3.

functional predicates, universal variables and logical operators, in particular the implication operator. These last two features enable the user to express rules. As reasoner we use EYE [7], a semibackward reasoning engine enhanced with Euler path detection. The main reason for our choice is the high performance of the reasoner. Existing benchmarks and results are listed on the EYE website [3].

3.2 OWL 2 RL in N3

To be able to support OWL 2 RL reasoning in N3 we used the OWL 2 RL/RDF rules as listed on the corresponding website [2]. Where possible, we made use of existing N3-translations of these rules as provided by EYE [4]. Missing concepts were added. Although the ACCIO ontology is designed for OWL-DL reasoning, the limitation to OWL RL had no impact for our specific use case.

To illustrate the idea of using rules for OWL reasoning, we give a small example: Listing 1 shows the class axiom rule[4] which is needed to deal with the rdfs concept `subclassOf`. For convenience we omit the prefixes in the formulas below. The empty prefix refers to the ACCIO ontology, `rdf` and `rdfs` have the same meaning as in Listing 1. Consider that we have the following T-Box triple stating that the class `:Call` is a subclass of the class `:Task`:

$$\texttt{:Call rdfs:subClassOf :Task.} \tag{1}$$

If the A-Box contains an individual which is member of the class `:Call`

$$\texttt{:call1 a :Call.} \tag{2}$$

an OWL DL reasoner would make the conclusion that the individual also belongs to the class `Task`

$$\texttt{:call1 a :Task.} \tag{3}$$

Our rule in Listing 1 does exactly the same: as Formula 1 and Formula 2 can be unified with the antecedence of the rule, a reasoner derives the triple in Formula 3. Other concepts can be handled similarly.

3.3 Decision Trees

The ACCIO ontology [5] provides the user with a huge variety of concepts which can, e.g., be used to describe patients (social background, needs, disease), staff

[4] The rule is the N3 version of the cax-sco rule in Table 7 on the OWL 2 Profiles website [2].

```
@prefix : <http://ontology/Accio.owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

{
  ?c rdf:type :Call.
  ?c :hasStatus :Active.
  ?c :madeAtLocation ?loc.
  ?p :hasRole [rdf:type :StaffMember].
  ?p :hasStatus :Free.
  ?p :closeTo ?loc.
}
=>
{
  (?p ?c) :assigned 200.
}.
```

Listing 2: Rule assigning a preference value to a staff member with status "free" who is close to the call-location.

members (skills, relationships to patients), and situations (locations of persons, states of devices). If all this information is actually available, decision trees can use all of it and be therefore quite complex. In this section we provide two simple rules which could be part of such a tree and we explain how these rules can be modified depending on the needs of an individual hospital.

Listing 2 shows a rule which, given an active call, assigns a staff member with a certain preference to that call. The EYE reasoner works with filter rules (queries), it is easy to search for the assignment of a staff member with the lowest or highest number. In our example, lower numbers mean higher preferences. The antecedence of the given rule contains certain constraints: the active call is made on a certain location and there is a staff member, who is currently free and close to that location. In such a case, our rule assigns the number 200 to the combination of call and staff member.

Listing 3 displays another rule: here, the reason of the active call is known. We have a staff member who has the required skills to answer that kind of calls, but this staff member is currently busy. Our rule assigns the number 100 to this combination of call and staff member. This means, in our current decision tree, we prefer this assignment to the one described by Listing 2.

Now, it could be, that another hospital has different priorities. Imagine for example that in this new hospital, no busy staff should be called if there is still a free staff member available, regardless of the reason of the call. We could easily adapt our decision tree by simply changing the assignment number of one of the rules. If we replace the triple

```
(?p ?c) :assigned 100.
```

in line 16 of Listing 3 by the triple

```
(?p ?c) :assigned 300.
```

```
@prefix : <http://ontology/Accio.owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

{
  ?c rdf:type :Call.
  ?c :hasStatus :Active.
  ?c :hasReason [rdf:type :CareReason].
  ?p rdf:type :Person.
  ?p :hasStatus :Busy.
  ?p :hasRole [rdf:type :StaffMember].
  ?p :hasCompetence [rdf:type :AnswerCareCallCompetence].
}
=>
{
  (?p ?c) :assigned 100.
}.
```

Listing 3: Rule assigning a preference value to a busy staff member who has the needed skills to answer the call.

the reasoner would prefer the assignment expressed by Listings 2 to 3.

Similarly, we can add extra conditions to the rules. Currently, the rule in listing 3 does not take the location of the staff member into account. We can change that by only adding the triples

```
?c :madeAtLocation ?loc. ?p :closeTo ?loc.
```

to the antecedence of the rule. To give this new rule a higher priority than the existing one, we would again only have to change the assigned number in the consequence. Rules with the same number are treated equally.

4 Results

We compared the reasoning times of our implementation with the results of our former implementation using the Pellet reasoner and subsequent SPARQL-queries. As a joint test set up we ran a sequence of events, which we list below, the expected reasoning result is indicated in brackets.

1. A patient launches a call (*assign nurse and update call status*)
2. The assigned nurse indicates that she is busy (*assign other nurse*)
3. The newly assigned nurse accepts the call task (*update call status*)
4. The nurse moves to the corridor (*update location*)
5. The nurse arrives at the patients' room (*update location, turn on lights and update nurse status*)
6. The nurse logs into the room's terminal (*update status call and nurse, open lockers*)
7. The nurse logs out again (*update status call and nurse, close lockers*)

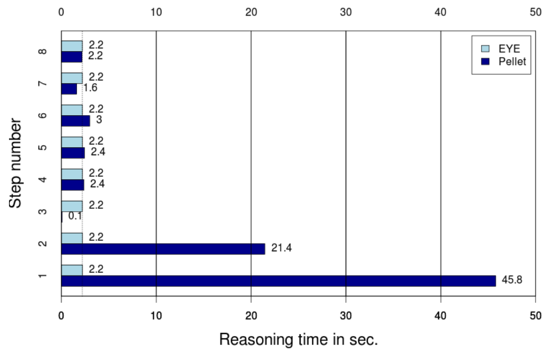
8. The nurse leaves the room (*update location and call status and turn off lights*)

We tested this set-up for two datasets, one dataset consisting of the nurses, hospital rooms and patients of one single ward, the other one for 10 wards. Figure 1a shows the difference in reasoning times of this use case on the same hardware settings[5], using two different technology stacks: the Pellet+SPARQL installation vs. the EYE installation[6]. The timings shown are the sum of all reasoning cycles.

As the figure shows, the reasoning times of EYE are almost one order of magnitude better than the reasoning times of Pellet+SPARQL. When we review the reasoning times per call for one ward (Figure 1b), we see that the EYE installation has far more predictable reasoning times, as it is more robust against more complex decision trees (e.g., the decision trees of the first two events are notably more complex than the other events' decision trees). Pellet+SPARQL is much faster than EYE in the third event, because this event does not trigger any reasoning for Pellet+SPARQL, however, a full reasoning cycle is performed by EYE. With an average reasoning time of about 2 seconds, the real-time constraint is achieved within small-scale datasets.

No. of	Time in sec.	
wards	EYE	Pellet
1	18	79
10	288	2 124

(a) Sum of reasoning times.



(b) 1 ward, reasoning time per event.

Fig. 1: Comparison of reasoning times. EYE is generally faster, and more predictable.

5 Importance and Impact

By using rule based reasoning instead of description logic based reasoning, we create a more performant system that is more easily configurable. The evaluation

[5] Debian "'Wheezy", Intel(R) Xeon(R) CPU E5620@2.40GHz, 12GB RAM

[6] Pellet 3.0 and OWL-API 3.4.5 vs. EYE 7995 and SWI-Prolog 6.6.6

shows that the current version does not meet the performance requirements to be applied for larger datasets, however, it can already meet the constraints in a small-scaled setting, it is on average faster than more traditional approaches such as Pellet+SPARQL, and it has more robust and predictable reasoning times.

The analysis of converting a decision tree into rules shows how a rule based reasoning method is more suited to implement decision trees, and how it is as such easier configurable to make changes to the implemented decision trees. The described analysis, being generic, can be used as a guideline for converting decision trees into rules for other use cases as well.

The results of this research are being supervised by Televic Healthcare, the leading eHealth electronics company in Belgium. This way, the chances that the findings of this research will be commercialized are quite high, and as such, potentially improve the workload of the nurses in a hospital significantly, as elaborated on in 1.

Further research will involve improving the automatic selection of relevant OWL-RL concepts. This way, reasoning over unused concepts is avoided, which, as we believe, will drastically improve average reasoning times, and more importantly, will make sure that the system will scale a lot better than it does now.

Acknowledgements The research activities described in this paper were funded by Ghent University, iMinds, the IWT Flanders, the FWO-Flanders, and the European Union, in the context of the project "ORCA", which is a collaboration of Televic Healthcare, Internet-Based Communication Networks and Services (IBCN), and Multimedia Lab (MMLab).

References

1. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A logical framework for the World Wide Web. Theory and Practice of Logic Programming 8(3), 249–269 (2008)
2. Calvanese, D., Carroll, J., Di Giacomo, G., Hendler, J., Herman, I., Parsia, B., Patel-Schneider, P.F., Ruttenberg, A., Sattler, U., Schneider, M.: OWL 2 Web Ontology Language Profiles (second edition). W3C Recommendation (Dec 2012), `www.w3.org/TR/owl2-profiles/`
3. De Roo, J.: Euler yet another proof engine (1999–2015), `http://eulersharp.sourceforge.net/`
4. De Roo, J.: EYE and OWL 2 (1999–2015), `http://eulersharp.sourceforge.net/2003/03swap/eye-owl2.html`
5. 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)
6. Parsia, B., Sirin, E.: Pellet: An OWL DL reasoner. In: Proceedings of the Third International Semantic Web Conference (2004)
7. Verborgh, R., De Roo, J.: Drawing conclusions from linked data on the web. IEEE Software 32(5) (May 2015), `http://online.qmags.com/ISW0515?cid=3244717&eid=19361&pg=25`