Semantic Web Reasoning by Swarm Intelligence

Kathrin Dentler
Dept. of Computer Science

Stefan Schlobach
Vrije Universiteit Amsterdam

Christophe Guéret
{kdr250,schlobac,cgueret}@few.vu.nl

ABSTRACT

Semantic Web reasoning systems are confronted with the task to process growing amounts of distributed, dynamic resources. We propose a novel way of approaching the challenge by RDF graph traversal, exploiting the advantages of Swarm Intelligence. Our nature-inspired methodology is realised by self-organising swarms of autonomous, light-weight entities that traverse RDF graphs by following paths, aiming to instantiate pattern-based inference rules.

Reasoning and Swarm Intelligence. It is widely recognised that new *adaptive* approaches towards *robust*, *scaleable* and *distributed* reasoning are required to exploit the full value of ever growing amounts of dynamic Semantic Web data. *Local* reasoning is an interesting option that supports decentralised publishing and has the potential to respect the provenance of the data and to allow users to keep control over their *privacy* and the ownership and dissemination of their information. Another advantage of decentralised reasoning is its capability to naturally integrate constantly changing data.

Adaptiveness, robustness and scaleability are among the main properties of swarms and can be attributed to the basic principles *lack of central control*, *locality* and *simplicity*. That is why the combination of reasoning and Swarm Intelligence can be a promising approach to obtain optimised reasoning performance by basic means. We introduce a swarm-based reasoning methodology and provide an initial evaluation of its feasibility and major characteristics. A model of a decentralised system is presented, which allows light-weight entities, that according to the employed swarm metaphor are referred to as beasts in the remainder, to traverse RDF graphs in order to calculate the deductive closure of these graphs w.r.t. the RDFS semantics. We investigate whether Swarm Intelligence can contribute to reduce the computational costs that the model implies, and make this new reasoning paradigm a real alternative to current approaches.

Methodology. In order to calculate the RDFS closure over an RDF graph G, a set of entailment rules has to be applied repeatedly to the triples in the graph. These rules consist of a precondition, usually containing one or two triples as arguments, and an action, typically to add a triple to the graph. The following is an exemplary rule:

rdfs3: If *p rdfs:range x .* and *s p o .* $\in G$ add *o rdf:type x .*

This process is usually done by indexing all triples and joining the results of two queries. With swarm-based reasoning we provide an index-free alternative for reasoning over large distributed dynamic networks of RDF(S) graphs. The idea is that swarms of light-weight beasts autonomously traverse the graph, each representing a reasoning rule, which might be (partially) instantiated. Beasts communicate only locally and indirectly. Whenever the conditions of a rule match the node a beast is on, it locally adds the newly derived triple. Our proposed paradigm envisages the Semantic Web as a connected collection of networks of data, which is constantly updated by beasts. In this set-up, only the active reasoning rules are moving in the network and not the data, minimising network traffic, as schema-data is typically far less numerous than instance-data. Given some added transition capability between graph-boundaries, our method converges towards closure. We claim that swarm-based reasoning is more adaptive and robust than other Semantic Web reasoning approaches, as recurrently revisiting beasts can more easily deal with added (and even deleted) information than index-based approaches.

Reasoning Model. Our beasts move through the graph by following its edges.[1] RDFS reasoning can naturally be decomposed by distributing complementary entailment rules on the members of the swarm, so that each individual is only responsible for the application of one rule. Therefore, we introduce different types of beasts, one type per RDF(S) entailment rule containing schema information.

Beasts are automatically instantiated by considering the schema information in the graph. If a concrete schema triple of a certain pattern is found, a reasoning beast is generated. Take for example the rule `rdfs3` for range restrictions: whenever in the schema an axiom `p rdfs:range x` is encountered, a beast of type $rb3$ is created with memory $\{p, x\}$. Table 1 lists some RDFS entailment rules, with the patterns that are to be recognised in column 2, and the generated beasts with their memory in column 3.

A beast $rb3$ is defined as a function $rb3$ (rb stands for reasoning beast) with memory $\{p, x\}$. Let us assume that while traversing a graph G, the instantiated range-beast arrives at node o from a node s via an edge (s, e, o). If $e = p$, it adds the triple `(o, rdf:type, x)` to G.[2] It moves on to a node n, where $(o, e_i, n) \in G$.

[1] There are alternatives such as random jumps to avoid local maxima, or guided jumps to locations where other beasts have been successful.

[2] Our beasts can walk both directions of the directed graph.

Rule	Pattern of schema triple	Beast: memory
rdfs2	p rdfs:domain x .	rb2: p x
rdfs3	p rdfs:range x .	rb3: p x
rdfs5	p_1 rdfs:subPropertyOf p . p rdfs:subPropertyOf p_2 .	rb7: p_1 p_2
rdfs7	p_1 rdfs:subPropertyOf p_2	rb7: p_1 p_2
rdfs9	c_1 rdfs:subClassOf c_2 .	rb9: c_1 c_2
rdfs11	c_1 rdfs:subClassOf c . c rdfs:subClassOf c_2 .	rb9: c_1 c_2

Table 1: Instantiation of reasoning beasts

Table 2 shows some beasts needed for RDFS reasoning with their pattern-based inference rules. Underlined elements correspond to the memory. Reasoning beasts $rb2$ and $rb3$ apply the semantics of `rdfs:domain` and `rdfs:range`, while $rb7$ and $rb9$ generate the inferences of `rdfs:subPropertyOf` and `rdfs:subClassOf`. We will refer to them as domain-beast, range-beast, subproperty-beast and subclass-beast.

Memory	If matches	Then add
rb2 : {p,x}	s p o .	s rdf:type x .
rb3 : {p,x}	s p o .	o rdf:type x .
rb7 : {p_1,p_2 }	s p_1 o .	s p_2 o .
rb9 : {c_1,c_2 }	s rdf:type c_1 .	s rdf:type c_2 .

Table 2: Inference patterns of reasoning beasts

The deductive closure C contains all triples that can be derived from the input data. Our method is sound, with the degree of completeness monotonically increasing over time.

Example. To illustrate the idea, let us consider two simple RDF graphs in Turtle about publications `cg:ISWC08` and `fvh:SWP` of members of our Department, maintained separately by respective authors and linked to public ontologies `pub` and `people` about publications and people.

```
cg:ISWC08
   pub:title "Anytime Query Answering" ;
   pub:publishedAs pub:InProceedings ;
   pub:author people:Gueret ;
   pub:author people:Oren ;
   pub:cites fvh:SWP .
fvh:SWP
   pub:title "Semantic Web Primer" ;
   pub:author people:Antoniou ;
   pub:author people:vanHarmelen ;
   pub:publishedAs pub:Book .
```

These two graphs denote two publications `cg:ISWC08` and `fvh:SWP` by different sets of authors. The graphs are physically distributed over the network and can be reasoned and queried over directly. Their information is extended with schema information:

```
pub:InProceedings rdfs:subClassOf pub:Publication
people:Person rdfs:subClassOf people:Agent
pub:author rdfs:range people:Person
```

Given the standard RDF(S) semantics one can derive that `cg:ISWC08` is a publication, and that the authors are also instances of class `people:Person` and thus `people:agent`. The formal semantics of RDFS and OWL enable the automation of such reasoning. Fig. 1 shows the RDF graph for the first publication. Red lines denote implicit links derived by reasoning.

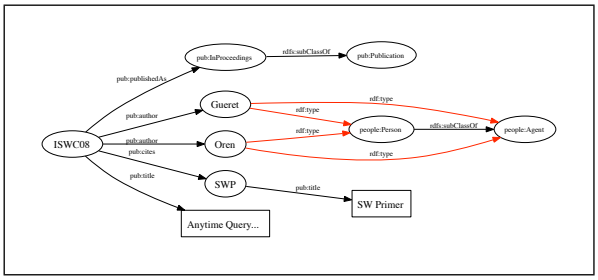

Figure 1: An RDF graph for our previous example

For the three schema axioms of our previous example, beasts are created. For the triple `people:Person rdfs:subClassOf people:Agent` a beast $rb9_1$ is created, which is instantiated with memory `people:Person` and `people:Agent`. (The other subclass-beast is generated accordingly.) For the range-triple `pub:author rdfs:range people:Person`, a beast $rb3$ is created with memory `pub:author` and `people:Person`. In our example, only one beast per instantiated type is created, in practise there will be more. The beasts are randomly distributed over the graph, say $rb3$ to node `fvh:SWP`, and similarly the other two beasts. Beast $rb3$ has now two options to walk. Moving to "SW Primer" will get it to a cul-de-sac, which means it needs to walk back via `cg:ISWC08` towards, eg. `person:Oren`. At node `person:Oren`, the walked path is `cg:ISWC08 pub:author person:Oren` which means $rb3$'s condition matches with its pattern, and it will add a triple `person:Oren rdf:type people:Person` to the graph. When, after walking other parts of the graph, the subclass-beast $rb9_1$ chooses to follow the `rdf:type` link from `person:Oren` to `people:Person`, it finds its memory condition matched, and adds `person:Oren rdf:type people:Agent` to the graph, and so forth.

Research questions. The price for our adaptive, index-free approach is redundancy: beasts have to traverse parts of the graph which would otherwise never be searched. The trade-off that needs to be investigated is thus whether the computational overhead of repeated exhaustive graph-traversal can be reduced by Swarm Intelligence so that the method offers both adaptive and flexible, as well as efficient reasoning.

Implementation and Experiments. We implemented our method based on the AgentScape platform, where each beast is implemented as an agent, and each distributed graph as an AgentScape dataprovider, linked with other dataproviders.

Findings. Our experiments had two goals: proof of concept, and to obtain a better understanding of the intrinsic potential and challenges of our new method. For the former, we run an RDF(S) reasoning system based on fully decentralised agents to calculate the semantic closure of a number of distributed RDF(S) datasets. From the latter perspective, the lessons learned were less clear-cut, as the results basically confirmed that tuning a system based on computational intelligence is a highly complex problem. However, the experiments gave us crucial insights in how to proceed in future work: most importantly on how to improve our attract/repulse methods for guiding swarms to interesting locations within the graph.