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 challenged to process growing amounts of distributed, dynamic resources. We present a nature-inspired method which is realised by autonomous entities that are traversing RDF graphs by following paths, aiming to instantiate pattern-based inference rules. We propose and implemented an approach that is based on swarm intelligence.

Swarm Reasoning

Recently, ever more distributed Semantic Web datasets with formal semantics are being published and interlinked. Most conventional reasoning engines focus on static, closed and consistent domains, whereas the Web is dynamic, open and everybody can state everything. In such an environment, completeness of inferences is not an option and correctness cannot be guaranteed. A second problem of Semantic Web reasoning is that the current way of publishing data is not ideal for ensuring **privacy and trust**. Decentralised publishing is an interesting alternative as it allows users to keep more control over their published data. Unfortunately, current reasoning methods mainly focus on batch-processing where all available information is loaded into, and dealt within, one central location. That works well for data that is collected centrally but is unsuitable for decentralised publishing. It is widely recognised that new **adaptive** approaches towards **robust** and **distributed reasoning** are required to process Semantic Web data. The aim of this poster is to introduce a reasoning method which is based on swarm intelligence and to provide an initial evaluation of its feasibility and major characteristics. We present a model of a decentralised, self-organising system, which allows autonomous, light-weight beasts to traverse RDF graphs and thereby instantiate pattern-based RDFS inference rules, in order to calculate the deductive closure of these graphs w.r.t. RDFS Semantics. We investigate whether swarm intelligence can help us to reduce the computational costs that our model implies, and make our new reasoning paradigm a real alternative to current approaches.

Method. 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 lightweight agents autonomously traverse the graph, each representing a reasoning rule, which might be (partially) instantiated. Whenever the conditions of a rule match the node a beast is on, it locally adds the newly derived triple. Given some added transition capability between graph-boundaries, our method converges towards closure. Our proposed paradigm envisages the Semantic Web as a connected collection of networks of data, which is constantly updated by beasts traversing this network. 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 instantiated by the schema 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 pattern that is to be recognised in column 2, and the generated beast with their memory in column 3.

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

Table 1: Instantiation of reasoning beast

[1]There are alternatives in form of teleportation, i.e. random jumps to avoid local maxima, or guided jumps to locations where other beasts have been successful.

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 $(o, e_i, n) \in G$.

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 <u>p</u> o .	s rdf:type <u>x</u> .
rb3 : {p,x}	s <u>p</u> o .	o rdf:type <u>x</u> .
rb7 : {p_1,p_2 }	s <u>p_1</u> o .	s <u>p_2</u> o .
rb9 : {c_1,c_2 }	s rdf:type <u>c_1</u> .	s rdf:type <u>c_2</u> .

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:subClass pub:Publication
people:Person rdfs:subclass 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.

For the three schema axioms of our previous example, beasts are created. For the triple `people:Person rdfs:subClass 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 for 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.

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

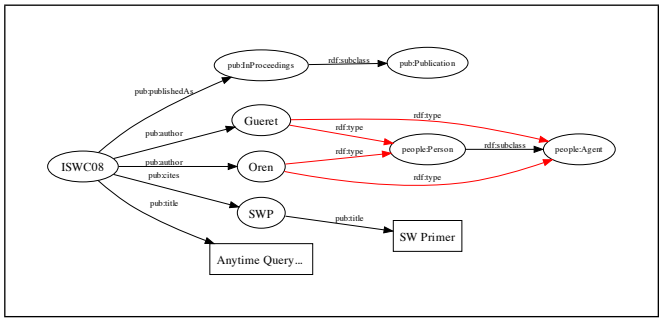

Figure 1: An RDF graph for our previous example

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 so that the method offers both adaptive and flexible, as well as sufficiently efficient reasoning. Our approach for solving this issue is based on swarm intelligence.

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 many other dataproviders.

Findings. Our experiments have 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 (physically) 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.