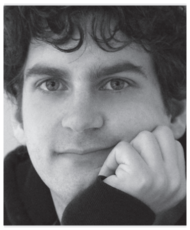
Editor: **Michiel van Genuchten**
VitalHealth Software
genuchten@ieee.org

Editor: **Les Hatton**
Oakwood Computing Associates
lesh@oakcomp.co.uk

Drawing Conclusions from Linked Data on the Web
The EYE Reasoner

Ruben Verborgh and Jos De Roo

This issue's installment examines a software program reasoning about the world's largest knowledge source. Ruben Verborgh and Jos De Roo describe how a small open source project can have a large impact. This is the fourth open source product discussed in the Impact department and the first written in the logic programming language Prolog. —*Michiel van Genuchten and Les Hatton*



THE WEB is the world's largest source of knowledge for people—and machines. In the beginning, those machines were mostly search engine crawlers that extracted keywords from natural-language texts. But now, the Web offers them something far more powerful: linked data.

Linked data goes back to the essence of the Web and information itself, by representing each piece of data as a link between two things. For example, Figure 1 shows a triple stating that Thomas Edison "knows" Nikola Tesla. Unlike most hyperlinks between Edison and Tesla, this one carries a specific meaning. Yet linked data's real benefit goes deeper: Edison and Tesla are represented by their Web address or URL. So, if you want to know more about Edison or Tesla, you can follow their URLs. Therefore, linked data is linked on two levels: each triple links two concepts, and those concepts link to more information about themselves.

If you look closely at Figure 1, you'll notice that the link type itself (the property) is also a URL. So, if a machine doesn't understand what "knows" means, it can look it up by following that URL. This principle is crucial to linked data: if you don't know something, look it up. Which Thomas are we talking about? What does "knows" mean? Follow the URL to find out.

If you follow the URL for this particular "knows" (http://xmlns.com/foaf/0.1/knows), you'll learn about the nature of this relationship. First, using "knows" means the involved subject and object are people. So even if we don't know Thomas or Nikola, we know they're people (as opposed to pets or cartoon figures). Second, this "knows" indicates reciprocity, so Nikola also knows Thomas. As humans, we can derive this without even being aware of who Nikola or Thomas are.

Such pieces of derived knowledge seem human-specific, but linked data

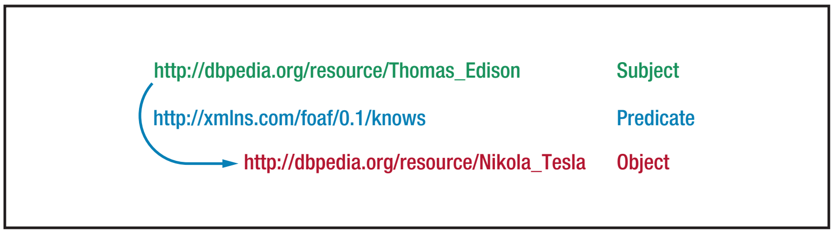

http://dbpedia.org/resource/Thomas_Edison
Subject
http://xmlns.com/foaf/0.1/knows
Predicate
http://dbpedia.org/resource/Nikola_Tesla
Object

FIGURE 1. At the heart of linked data are triples: predicates link a subject to an object. This triple states that Thomas Edison "knows" Nikola Tesla.

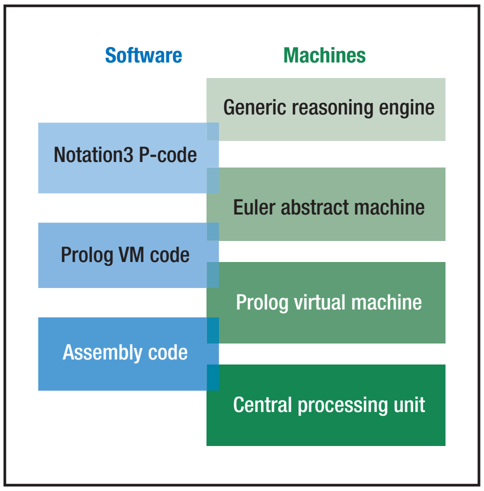


FIGURE 2. The EYE stack offers a generic reasoning engine on top of interchangeable virtual machines.

also lets software reasoners arrive at the same conclusions. This column discusses how the EYE reasoner can do just that and how industry is already using this.

Reasoning on the Semantic Web with EYE

Since 2006, EYE (available under the W3C license at http://eulersharp.sourceforge.net) has been tackling such reasoning challenges on a large scale, thereby forming a part of the bigger Semantic Web vision. In this vision, machines perform tasks on the Web for people, combining linked data with concepts such as reasoning and proof to turn that data into knowledge and concrete actions.

Continuing the previous example, let's see how EYE moves from data to conclusions. We can describe the `knows` property with existing concepts such as `domain`, `range`, and `SymmetricProperty`. Many reasoners have built-in knowledge; they would know what those concepts mean and how to apply them. The drawback, of course, is that only built-in concepts can create new knowledge. So, EYE was designed to have the least amount of inherent knowledge; it's extensible through rules so that new knowledge can be created. For example, the EYE webpage offers rules that model the `SymmetricProperty` concept as follows:

```
{
  ?property rdf:type owl:SymmetricProperty.
  ?subject ?property ?object.
}
=>
{
  ?object ?property ?subject.
}.
```

Perhaps surprisingly, this rule is also a (special) triple: antecedent–then–consequent. It explains that, if a symmetric property links a subject to an object, then it also links that object to the subject. So, if we give EYE the Edison–knows–Tesla triple, the description of "knows," and the previous rule, EYE will derive that Tesla also knows Edison.

In isolation, this might not look spectacular. After all, we gave EYE all the knowledge it needed. However, the rules can cascade at a large scale, so it becomes interesting if we give EYE tons of knowledge—say, millions of triples—and it derives just the facts we want. For instance, we might ask for "contemporaries of Nikola Tesla," and by the (derived) fact that Tesla knows Edison, EYE could find Edison as an answer. In a different use case, we might feed EYE with medical information of millions of patients to identify adverse drug effects that previously were hidden. EYE is used regularly to solve such real-world clinical applications.

EYE's Internals

Algorithmically speaking, EYE is a theorem prover. Users set a goal, and EYE tries to reach it by applying logical rules similar to what we mentioned previously, mostly working backward from the goal. To evade endless loops, the algorithm avoids needlessly repeating previous work through Euler path detection, similar to Leonhard Euler's Königsberg bridge problem. EYE interprets each logical rule $P \Rightarrow C$ (where P is a precondition and C is a consequent) as P AND NOT(C) $\Rightarrow$ C, so that rules execute only when they can generate new triples.

A key characteristic of EYE's architecture is portability, because interoperability is crucial to the Semantic Web. EYE components are interchangeable (see Figure 2). EYE runs in a Prolog virtual machine, which is compiled to assembly code and runs directly on the CPU. The core of EYE, the Euler Abstract Machine, is compatible with at least two major Prolog engines (YAP and SWI-Prolog). This machine accepts N3 (Notation3) P-code, which is a Prolog representation resulting from parsing RDF (Resource Description Framework) triples and N3 rules. The entire stack thereby becomes a generic reasoning engine, which is extensible with any kind of domain-specific rules. Because EYE can output N3 P-code to a file, users can create specific reasoning-engine instances that have a certain rule set preloaded for a particular domain.

EYE also exhibits external compatibility: it accepts exchangeable N3 and Turtle RDF documents from any source on the Web. Users can ask EYE to generate a proof explaining how the given goal is reached. Such proofs use a publicly available, interoperable vocabulary, so other parties can follow and understand the line of reasoning. Most important, this mechanism allows independent proof validation, which contributes to one of the forms of trust on the Semantic Web. If a certain party comes to a conclusion, any other party can thus verify why that conclusion is valid. Furthermore, EYE is compatible with the W3C's Rule Interchange Format (RIF), so the rules that constitute deductions and proofs can be exchanged even beyond the N3 language.

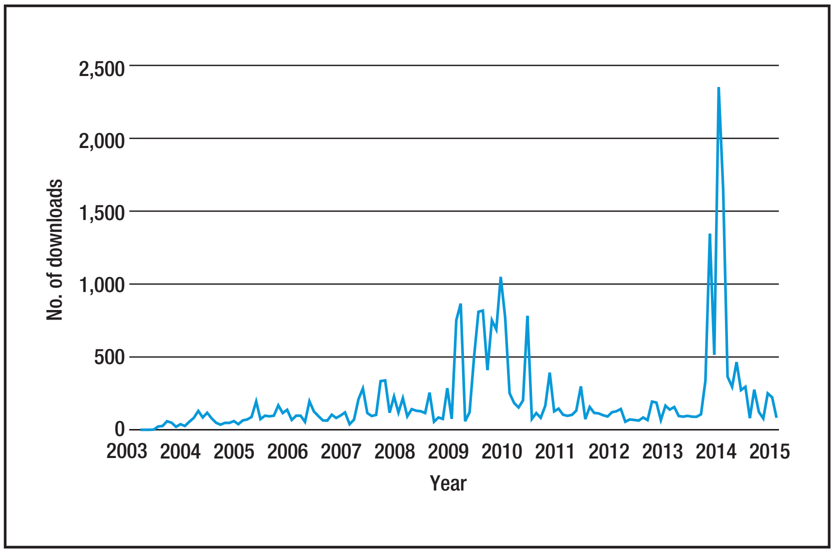


FIGURE 3. EYE has been downloaded 30,000 times, at an average of 200 downloads per month.

EYE's Development and Use

EYE originated around 1999, the beginning of the Semantic Web era, as a program called Euler. After several iterations in other programming languages, the Prolog incarnation started in 2006, becoming the predominant and only implementation, aptly named EYE (first for Euler YAP [Prolog] Engine, then for Euler Yet Another Proof Engine). Since then, more than 150 updates of EYE with new features and bug fixes have been released; the schedule was recently adapted to a three-month cycle. The number of downloads from SourceForge has steadily increased, with an average of 200 per month and a total of 30,000 (see Figure 3). EYE is also available as a public Web service and a Docker image (with 500 downloads so far).

Jos De Roo develops and maintains EYE, and 30 people in its community support it through bug reports, which have totaled more than 200 so far. These reports are usually addressed within days. EYE's current code base consists of roughly 10,000 lines of Prolog source code, which amounts to 100,000 lines of Prolog virtual-machine code.

Being a generic reasoner, EYE is applicable in a range of contexts. In essence, it can tackle any problem domain modeled in RDF (and N3 for rules). In particular, if a domain was modeled with existing core ontological concepts from the Semantic Web, such as those from the RDFS (RDF Schema) and OWL vocabularies, EYE can derive new triples right away because it can reuse the existing N3 rules for these concepts.

In particular, EYE has several active applications in the medical sector.[1] One example is the SALUS (Scalable, Standard Based Interoperability Framework for Sustainable Proactive Post Market Safety Studies) project, for which Agfa Healthcare is developing a semantic interoperability service for postmarket drug safety studies. Different data sources each employ different ontologies, and EYE lets users transform queries and data to and from CREAM (Clinical Research Entities Patterns: Advanced Model). This works through rules for mapping each ontology to CREAM. The Paris public hospital system is employing EYE for similar use cases.

RELATED WORK IN REASONERS

A few alternatives to the EYE reasoner exist—most notably, the cwm, Jena, and Fuxi reasoners. However, EYE offers far superior performance. For instance, EYE solves the Deep Taxonomy Benchmark problem (http://eulersharp.sourceforge.net/2003/03swap/dtb-note) for 100,000 triples in 4.8 seconds, whereas cwm needs nine days and Jena goes out of memory. Researchers have reported similar drastic performance improvements with the RESTdesc Composition Benchmark (http://eulersharp.sourceforge.net/2003/03swap/rcb-note) and Basic MONADIC Benchmark (http://eulersharp.sourceforge.net/2003/03swap/bmb-note).

Three other products described previously in *IEEE Software*'s Impact department are similar to EYE. RealPlayer is also an open source product that has seen industrial applications.[1] It's much larger (millions of lines of code). In 2010, the community comprised 150,000 engineers, many of whom worked in mobile phone companies that were part of the Helix community. YAWL (Yet Another Workflow Language), another open source product, was also written by a small team of engineers and is comparable in size and installed base.[2] Bayesian networks are similar to EYE in terms of scientific grounding, the availability of a free version, size, and volume.[3]

References

1. L. Bouchard, "Multimedia Software for Mobile Phones," *IEEE Software*, vol. 27, no. 3, 2010, pp. 8–10.
2. M.J. Adams, A.H.M. ter Hofstede, and M. La Rosa, "Open Source Software for Workflow Management: The Case of YAWL," *IEEE Software,* vol. 28, no. 3, 2011, pp. 16–19.
3. N. Fenton and M. Neil, "Decision Support Software for Probabilistic Risk Assessment Using Bayesian Networks," *IEEE Software*, vol. 31, no. 2, 2014, pp. 21–26.

Even though the use of reasoning isn't always obvious, EYE is at the heart of several applications. Ghent University and iMinds are using EYE's proof capabilities to solve the problem of semantic service composition.[2] Each service's functionality is expressed in the RESTdesc language, consisting of N3 rules, so that EYE can reach a goal with them—even though they're not static data rules. Agfa Healthcare experiments with RESTdesc rules for semantic clinical workflow composition and execution. In this kind of composition, data, (regular) rules, and services work seamlessly together. So, regular ontological reasoning can take place at the same time as service composition, creating service connections that were previously infeasible.

For a look at technologies similar to EYE, see the sidebar.

EYE's Future

The Semantic Web world is still under development, but the linked data principles have already inspired many people to make their data available in RDF format. EYE's role there is clear: ensuring that linked data in one vocabulary can be easily transformed into another, thanks to explicit relations between those vocabularies. Because these building blocks are mostly in place, the EYE project will focus on *unifying logic* and *explainable reasoning*.

Unifying logic designates a logic framework that can be shared across different agents on the Web. So far, Semantic Web logic has been fragmented. N3 hasn't been standardized, but the Turtle syntax on which it's based was finally standardized in 2014. In practice, not everything can be expressed elegantly on the level of N3 itself. Some basic blocks, such as mathematical addition or string comparison, are best exposed through built-ins, special predicates that aren't necessarily implemented in N3. To communicate with each other, agents must agree on a predefined set of built-ins with a rigorously specified meaning. If they don't, results and corresponding proofs will be untrustworthy. Although the RIF standard includes such a set, adoption has been minimal. EYE and other reasoners must strive to improve this.

Explainable reasoning indicates the possibility of agents producing and exchanging proofs. Any conclusion reached by a reasoner must be independently verifiable to enable real-world applications. For instance, for financial or safety calculations, all the involved parties want to ensure they can trust the result. And, if these proofs are used in workflow contexts, they reflect on an entire agent-driven process. EYE wants to play a role in such an agent-enabled Web, and the current proof-enabled reasoning functionality is the first step toward that.

In the future, reasoners could have a large impact on software. A few decades ago, software took over many jobs from hardware: calculations that used to be hardwired became soft-wired as lines of code. Although some highly optimized calculations still happen in hardware, most are now performed with special-purpose software on generic-purpose hardware. Similarly, reasoners will start rewiring software for specific situations. EYE already does this when performing rule-based Web service composition. In the end, this opens the door to automatically customized software processes. What software once did for hardware, reasoning might one day do for software.

References

1. E. Papageorgiou et al., “Application of Probabilistic and Fuzzy Cognitive Approaches in Semantic Web Framework for Medical Decision Support,” *Computer Methods and Programs in Biomedicine*, vol. 112, no. 3, 2013, pp. 590–598.
2. R. Verborgh et al., “Capturing the Functionality of Web Services with Functional Descriptions,” *Multimedia Tools and Applications*, vol. 64, no. 2, 2012, pp. 365–387.

RUBEN VERBORGH is a postdoctoral researcher in semantic hypermedia at the Multimedia Lab, a research group of iMinds and Ghent University. Contact him at ruben.verborgh@ugent.be.

JOS DE ROO is a researcher at Agfa Healthcare. Contact him at jos.deroo@agfa.com.