Semantic Web Support for Intelligent Search and Retrieval of Business Knowledge

Valentina Tamma, *University of Liverpool*

In today's Web, information is primarily intended to be read and processed by humans; it can't be readily comprehended and manipulated by agents. The intelligence underlying search tasks, as well as the assessment of the retrieved pages' relevance, comes mainly from human sources, with limited support from software.[1] Although this type of processing is still adequate for searches returning a few hundred pages, it can't scale to the volume of information available in business, where enterprises couple the vast amount of data available on the Web with company documents and databases. Current keyword-based search engines can't fully capture the intrinsic richness of natural language; synonymy and polysemy, for example, pose difficult problems for a keyword-based search task. Enhancing search engines with lexicons such as WordNet[2] can help relieve these problems, but this doesn't identify and resolve more complicated types of ambiguity. Furthermore, keyword-based search engines make little provision for the formulation of very specific queries, particularly those that make use of relationships between entities.

Semantic Web technologies offer a possible way to overcome these limitations. The Semantic Web is an evolution of the current Web that represents information in a machine-readable format, while maintaining the human-friendly HTML representation.[3] In the Semantic Web, ontologies give resources shared, machine-processable meaning by modeling the entities and processes used to describe both the content of a Web resource and, more importantly, the logical relations between the resources.[4] Ontological models allow the *annotation* of Web documents (modeling the representation of information contained in them) and thus the formulation of more precise queries to retrieve documents. Annotation normally involves creating metadata items (as instances of concepts from the ontology) to represent specific entities recognized in the resources, and then linking this metadata to the resource as its description. Many research efforts have thus focused on providing automatic or semiautomatic ways to annotate Web documents in various formats—mainly text, but also structured formats such as databases.

In this article we present our experience developing QuestSemantics (QS), an agent-based platform that uses fine-grained business knowledge to support semiautomatic discovery, annotation, filtering and retrieval of information resources on the Internet and in intranets. We designed QS to maximize the separation between the different types of knowledge represented—domain- versus task-specific knowledge, and application versus generic knowledge. The goal of this separation is to achieve reusability and easy customization of the platform's various agents, thus allowing semantics-based search in various task and domain scenarios. QS includes two main components:

- a general framework for (semi-)automatic resource annotation based on a detailed ontological model of the domain and
- a user-friendly search interface that allows the formulation and execution of knowledge-based queries over the generated metadata.

We designed QS for application scenarios that exploit different information sources to provide searchable knowledge. The process often differs only slightly between different application scenarios and different domains. The aim of the general framework for annotation is to abstract from different scenarios all the common implementation and policy details in order to reduce and simplify application-specific code.

Knowledge-Independent Components

Because QS is intended as a generic platform, we designed its components to be customizable to the specific domain of application. Therefore, a main concern in designing the platform was to limit its customization to domain-related aspects only. The design of QS distinguishes between domain knowledge and task knowledge. *Domain knowledge* describes all relevant entities in a specific domain of knowledge, representing a state of affairs and constraining the possible states it can evolve into. *Task knowledge*, in general, uses domain knowledge to describe relevant entities with respect to the required tasks.[5]

The only decisions that QS makes at platform level relate to the formalisms adopted for representing domain and task knowledge. A domain ontology needs a formalism that allows easy expression of taxonomical and nontaxonomical relationships among agents—static knowledge. A task ontology, on the other hand, must represent dynamic operations such as sequences, selections, and iterations that are necessary to represent tasks. The Semantic Web standard for representing ontologies is the Web Ontology Language, OWL.[6] Although OWL is adequate for modeling domain knowledge, it isn't suitable for representing dynamic operations. For these, we supplement OWL ontologies with rules represented using the Semantic Web Rule Language, SWRL.[7,8] Such an extension is necessary, for example, to express part-whole relations;[9] description logic, the representation formalism underlying OWL, isn't sufficiently expressive to formalize these relations. QS represents procedural knowledge, on the other hand, by mixing declarative rules with a traditional programming language (Java). It then represents tasks using clauses—a set of conjunctive premises and a single consequence, with the consequence represented by a block of executable code.

A task ontology must represent dynamic operations such as sequences, selections, and iterations that are necessary to represent tasks.

Annotation and Search

The QS framework consists of two stages. In the *annotation stage*, QS uses both domain knowledge and task-specific knowledge (such as layout specification, annotation, and filter rules) to create semantic metadata about the information sources. It then uses this metadata in the *search stage*, in which it answers specific queries from the user, using domain knowledge to guide the query process.

The annotation stage involves a collaboration of several agents, each of which provides distinct process capabilities:

- harvesting live information sources, ensuring retrieved information is up to date with the latest information available;
- analyzing the retrieved resources using knowledge encoded in the heuristic task rules to identify which are of interest for the annotation component;
- annotating the analysis results using domain ontologies, identifying instances of concepts, and, where possible, retrieving attributes and stating relations between instances; and
- storing the metadata resulting from the annotation process in a Resource Description Framework (RDF) database (www.w3.org/RDF).

The search stage is primarily devoted to retrieving specific information from the metadata stored in the last step of the annotation phase. QS expresses queries in the SPARQL query language[10] and uses the ontology representing the application domain to impose constraints on potentially matching resources. Responses to queries are lists of matching resources, containing the metadata descriptions and a pointer to the original source (for example, a Web page or a database record set). A graphical search interface enables user specification of the semantic queries in an intuitive, nontechnical manner and allows clear presentation of and access to the resulting resources.

Framework Design

The framework design (Figure 1) is based on two main components implemented as multiagent systems: the annotation engine[11] and the search engine.[12]

An annotation engine analyzes and filters the retrieved documents (the annotation stage), and a semantic search engine provides fine-grained access to the filtered documents (the search stage). The two components share a store component, which stores all data: document contents, ontologies and metadata instantiations, and intermediate results created by the analysis and annotation components.

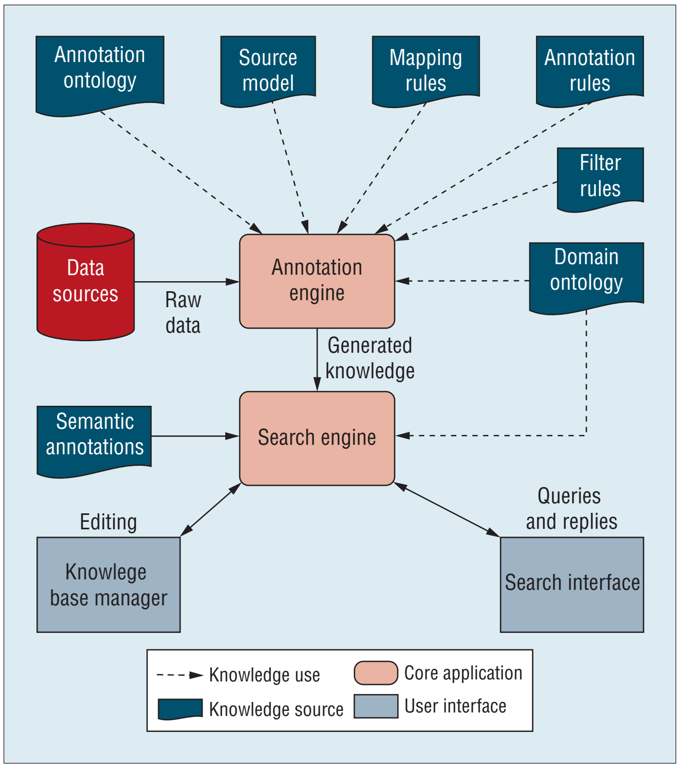

Figure 1. QuestSemantics (QS) general system architecture. The annotation engine applies mapping and annotation rules to the database. The knowledge base manager reviews and edits annotations. The search interface is a Web-based GUI for creating queries and viewing results. The search engine answers semantic queries over annotated resources.

Annotation Engine

The annotation engine retrieves documents from their sources and then analyzes, annotates, and filters them on the basis of the application needs. Each of these functions is performed by a specific element that represents an implementation of one of the interfaces (harvester, analyzer, or semantic annotator). At this level of abstraction, QS separates task-specific knowledge and domain knowledge: The analyzer element poses only the task-specific knowledge available—for example, how to find relevant information on a Web page. The semantic annotator element uses domain knowledge to create the actual metadata. We obtain these independent components by leveraging the distinction between the knowledge needed for each functionality, so that changes in task or domain have an impact on only one component. Moreover, confining the task-specific knowledge to the analyzer system makes the search component completely independent of the way information is retrieved, easing the process of using multiple knowledge bases to answer users' queries. Now we'll look at the elements of the annotation engine component in more detail.

Harvester. An implementation of the harvester interface must be able to retrieve information resources and convert them into a form suitable for the annotation process. In the case of Web pages, the harvester retrieves the pages and saves them in the store component as text documents. When the source is a database, the harvester retrieves first the database schema and then the contents and saves them in XML format.

Analyzer. Analyzer elements define methods to extract relevant information from an input information source and store it in an intermediate format suitable for the annotation engine. Figure 2 shows its architecture.

Document-layout-specific information is encoded in the form of regular expressions (or with specialized Java code) into an implementation of the MatchingPattern interface. A parser implementation uses a set of these implementations, and a parser together with its MatchingPattern elements forms a rule. Rules are considered atomic objects, meaning that the relevant information found by the MatchingPattern elements within a rule are extracted only if the input document or source satisfies all the MatchingPattern elements; in this case, the rule is *applicable*. Some rules can condition the applicability of other rules—for example, if one rule determines that the current resource is unsuitable, it forces all subsequent rules to be skipped (a *blocking rule*).

Semantic Annotator. This element creates the RDF models representing the information highlighted by the analyzer, building source metadata according to the domain- and application-specific ontologies. Figure 2 shows its architecture. Analogously with the internal structure of the analyzer element, annotation takes place through AbstractDocumentMatchingPattern implementations. Each implementation extracts a specific piece of information from the analyzer output, and annotator processes create and formalize the metadata into an RDF model. Annotators and AbstractDocumentMatchingPatterns are grouped into AnnotationRules, which can be blocking or nonblocking. The semantic annotator element is the first point in the process where the form of the source information becomes unimportant—that is, it is agnostic with respect to whether the data originates from Web pages or from other sources, such as a database. Filters in the semantic annotator apply predefined filter rules to determine whether a specific resource is suitable for use by the search engine. For example, a filter might remove information that is no longer up to date or useful (such as information that expires after a certain amount of time).

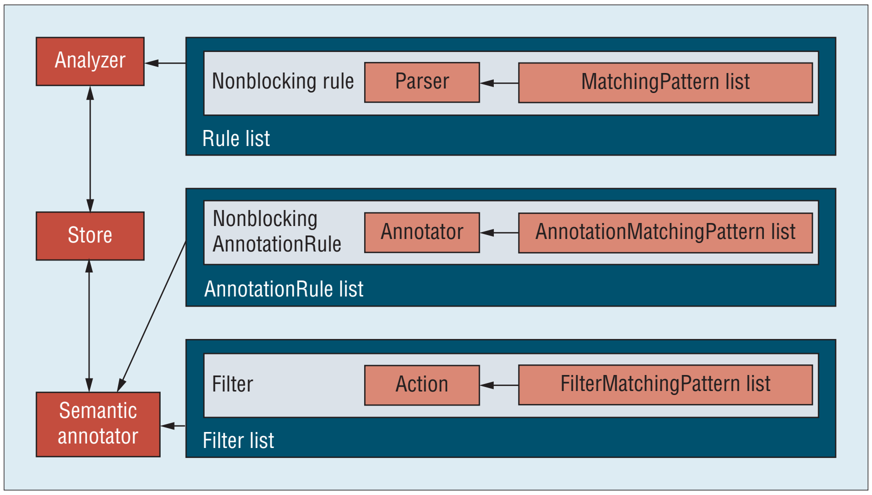

Figure 2. QS annotation components detailed architecture.

Store Element. Each step of the annotation process produces data that must be saved persistently, both for performance (for example, to save retrieved documents so that they are available for the analysis step) and to keep track of connections between information items, such as the source of a specific annotation. The store interface enables an application to save and retrieve data identified by a URI, such as byte streams (typically containing text documents such as HTML pages), Java maps containing intermediate mapping results, and RDF models containing finished annotations. In addition, the store interface can save relations such as the fact that a specific URI is an alternate name for another equivalent resource—that is, in OWL terms, the two resources are `owl:sameAs`. This is particularly useful when different documents describe a single conceptual resource. The annotation rules retrieve all available information for the resource, addressing the problem of information that is logically related but physically disconnected.

Semantic Search Engine

The framework's search engine component queries the information generated by the annotation component. It accepts queries posed in SPARQL and returns a set of links to matching resources. A specialized search interface lets users develop an abstract model of a semantic query, pose it to the engine, and then review the resulting matched documents. The search interface gives end users (people who aren't experts in Semantic Web technologies) a way to access the resources filtered and annotated by the semantic annotator component. It is also possible to add and delete entities and properties (with related values), so that a user can interact with the knowledge base to fine-tune the query, making subsequent searches more accurate.

The key aim for the query interface is to give the user an intuitive and clear abstract query model that hides, as much as possible, the underlying complexity of representation and reasoning.

Furthermore, the agents in the search engine multiagent system exhibit various autonomic features that aim at making the system more robust and scalable.[12]

The QS system has been deployed in two different commercial test cases in the UK. In the first case, QS was used to examine specific Web-published documents for commercial opportunities matching the business interests of the customer company. In the second deployment, QS was used to perform knowledge-based searches over existing database sources. In evaluating the performance of the search system in both applications, we could see that by using ontological knowledge and ontology-based annotations, users could perform more accurate queries while being returned up to 71 percent fewer documents than with a keyword-based search engine—in the best cases eliminating more than 90 percent of the irrelevant documents.[11] We are now in the process of further refining these two deployments, and we are planning more industrial deployments in the near future with other UK companies.

References

1. V. Uren et al., "Semantic Annotation for Knowledge Management: Requirements and a Survey of the State of the Art," *J. Web Semantics,* vol. 4, no. 1, 2006, pp. 14–28.

2. G.A. Miller, "Wordnet: A Lexical Database for English," *Comm. ACM*, vol. 38, no. 11, 1995, pp. 39–41.
3. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific Am.*, vol. 284, no. 5, May 2001, pp. 34–43.
4. R. Studer, R. Benjamins, and D. Fensel, "Knowledge Engineering: Principles and Methods," *J. ACM*, vol. 25, nos. 1–2, 1998, pp. 161–197.
5. G. van Heijst, A.T. Schreiber, and B.J. Wielinga, "Using Explicit Ontologies in KBS Development," *Int'l J. Human-Computer Studies*, vol. 46, no. 2, 1997, pp. 183–292.
6. D.L. McGuinness and F. van Harmelen, eds., *OWL Web Ontology Language Overview*, W3C recommendation, 2004; www.w3.org/TR/owl-features.
7. I. Horrocks et al., "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," W3C member submission, 2004; www.w3.org/Submission/SWRL.
8. I. Horrocks et al., "OWL Rules: A Proposal and Prototype Implementation," *J. Web Semantics*, vol. 3, no. 1, 2005, pp. 23–40.
9. M.E. Winston, R. Chaffin, and D. Herrmann, "A Taxonomy of Part-Whole Relations," *Cognitive Science*, vol. 11, no. 4, 1987, pp. 417–444.
10. *SPARQL Query Language for RDF*, W3C recommendation, 2008; www.w3.org/TR/2008/REC-rdf-sparql-query-20080115.
11. I.W. Blacoe et al., "QuestSemantics—Intelligent Search and Retrieval of Business Knowledge," *Proc. 18th European Conf. Artificial Intelligence* (ECAI 08), IOS Press, 2008, pp. 648–652.
12. I.W. Blacoe, V. Tamma, and M.J. Wooldridge, "Evaluation of Scalable Multi-Agent System Architectures for Searching the Semantic Web," to be published in *Int'l J. Metadata, Semantics and Ontologies*, 2010.

Valentina Tamma is a lecturer in the Department of Computer Science at the University of Liverpool, where she coordinates the Semantic Web Lab in the Agent ART Group. Her research focuses on ontologies in open and distributed environments, such as multiagent systems, the Semantic Web, and grid systems. She holds a PhD in ontologies and information systems from the University of Liverpool, UK. Contact her at v.tamma@liverpool.ac.uk.