

Fast semi-automatic generation of ontologies and their exploitation

[Abstract: Ontologies can be used to enhance information retrieval and rendition greatly. They also enhance the machine readability and understandability of web documents. Ontologizers and ontology representation languages respectively, can efficiently deduce and describe relationships among information from their metadata in ventures like the Semantic Web. Complete automatic deduction of ontologies from a set of definitive natural language documents, for a specific domain, has not yet been achieved. We describe LOGS, Lightweight universal Ontology Generation and exploitation architecture, which takes a step toward that direction and the fast, lightweight approach used in it. We also describe the refinement process, heuristics and the new algorithm used in the generation step of LOGS. We illustrate the use of LOGS in 'Eagle', a comprehensive ontology management and information retrieval tool for web documents. Eagle allows the generation and ratification of domain ontologies, and demonstrates their exploitation in intelligent information retrieval and in intelligent interfaces for presenting the information. The tool and LOGS are highly scalable and the ontologies generated can be easily integrated with other tools and domain ontologies, as we use popular ontology languages for representation and processing. Finally, we show how LOGS could be applied in other domains.]

Introduction:

In its general meaning, **Ontology** is the study or concern about what kinds of things exist - what entities there are in the Universe. An Ontology is the working model of entities and interactions in some particular domain of knowledge or practices, such as electronic commerce or planning. An ontology, is a set of concepts - such as things, events, and relations - that are specified in some way, such as specific natural language, in order to create an agreed upon vocabulary for exchanging information as much as terminology is used to assign names to various object, places, concepts etc. An Ontology is also defined as shared formal conceptualisation of a particular domain. In other words, "An ontology is a specification of a conceptualisation" -[Gruber 93]. Ontology, developed in Artificial Intelligence to facilitate knowledge sharing and reuse, could become a possible solution [Frensel] to eliminate these negative aspects of Information Retrieval from different perspectives[Frensel 2]. First, it can act as the link between users and information by logically abstracting the information so as to provide the concepts and relations (explicit semantic representation of knowledge) for users to form and refine their queries consistently. Second, it retrieves relevant information based on its inference functions, which could really fulfill the term "intelligent information retrieval [Yee]". Ontologies are considered taxonomic hierarchies of classes, class definitions, and associations. Ontologies need not be limited to these forms or to traditional dictionary type definitions of terms, which convey only the term, unlike an encyclopaedia entry which can be informative as to specify a conceptualisation one needs to state axioms that do constrain the possible interpretations for the defined terms. Ontologies have been used in various areas, in AI, such as in intelligent agents, decision support, retrieval, e-commerce, enterprise modelling etc.

Automatic construction of domain ontologies from text documentation is as yet an open question. There have been significant attempts to construct narrow domain ontologies, semi-automatically. The use of ontologies in data mining and knowledge based systems and agents,

is well studied and is applied in industry as well. We describe our Lightweight Universal Ontology Generation and Exploitation architectures (LOGS), which takes a step toward generating ontologies automatically and using them in intelligent interface construction and intelligent querying. The rest of this paper is organized as follows. We briefly discuss our approach and outline the architecture of LOGS. We then describe the sample application Eagle, and its uses. We discuss its performance and make some comments on using LOGS in integrating a local site with a Semantic web type application. Finally, we mention limitations of LOGS and scope for further work.

Automatic Construction:

Large ontologies are required for understanding unrestricted natural language text and merging and integrating and exploiting knowledge bases from different sources. Handcoding techniques are predictably obsolete, inflexible, and inappropriate for large-scale ontology development and some of the tasks have been successfully automated. [Sowa].

What exactly does comprehensive automatic extraction or construction of ontologies entail? The first expectation is that it be an end-to-end solution, from generation to exploitation and retrieval. If a domain expert has identified a set of definitive documents on any topic, written in natural language, which describe most of the central concepts, could we construct, without the use of tools such as Protégé or OntoEdit, or the services of the domain expert or extensive labor as in Yahoo! , we should be able to generate an ontology connecting the concepts we identified and be able to machine process it for exploitation. The documents that we use must be defining in the sense that there must be a clear central idea, with attendant secondary foci, much as the Group theory test document supplied with the tool describes Groups, but also holds forth on semi-groups, monoids, cyclic groups, abelian groups, sets and monoids as well, albeit in passing. It must follow some the clear style and format guidelines supported under LOGS which are listed elsewhere.

As LOGS is based on observed patterns and structure, and adapts to both user and subject, it is faster than similar systems.

Ontology Representation:

Various notations are available to us, for both, pictorial and textual rendition of ontologies. Ontolingua and Loom are examples of representational languages using First Order Logic. When needing to extend ontology exchange to the web, we realise a need to define representational languages over the existing HTML syntax, so that they are machine readable. SHOE (Simple HTML Ontology Extensions) was one such effort. With the recent emergence of XML, languages such as XOL, OML and RDFS, the Resource Description Framework Schema language etc have been defined over XML. All of these use various kinds of additional tags over existing XML syntax. OIL extended RDF and RDFS. "OIL unifies the epistemologically rich modelling primitives of frames, the formal semantics and efficient reasoning support of description logics and mapping to the standard Web metadata language proposals." In LOGS, we use DAML+OIL, due to its growing popularity, existence of a knowledgeable user community and its close alliance with the Semantic Web projects. DAML

stands for Darpa Agent Markup Language and OIL stands for Ontology Interchange Language. These independent ventures, from either side of the Pond, were combined to give the current DAML+OIL notation. DAML+OIL provides a rich set of constructs with which to create ontologies and to markup information so that it is machine readable and understandable. It allows the easy merge, import of external ontology descriptions and extensions and is object-oriented. It also has a fair user group and is an attempt to merge the best features of its immediate predecessors and is easy to use. It has additional features such as bounded lists, inheritance, formal semantics, inference, local restrictions etc. This caused our choice of representation language, although any HTML based ontology language can be plugged into our solution and in our implementation we tap only a very small part of the language, by design. DAML+OIL is applied in ventures like Ontoweb and Wonderweb.

LOGS:

As noted earlier, complete automatic generation being infeasible, and most automation being very domain restricted, such as within medical diagnosis or a corporate intranet, an ontologizer which is domain-insensitive becomes useful. [Zhu] However, it becomes too complex to implement, as the treatment and exploitation across various domains, such as medicine and commerce, above, might be completely different. For example, in the former, one might only be interested in associations that are not obvious, whereas, in the latter, deducting trends such as user behavior from the meta-data in the ontology may be more useful. Therefore, we adopted a two-pass solution to the problem. We designed the LOGS architecture, which contains all the functionality, excepting the structural and semantic patterns for input and the methods needed for exploitation such as the search algorithms and final user interfaces. All other functionality is contained in LOGS and described below. When we need to develop an end-to-end solution, we provide the input and interface domain information needed, a training corpus for the general domain, and customize LOGS through the Application Programmer Interface to generate the final application. This application, including an ontological server, can then be deployed in various ways.

LOGS architecture consists of the following modules: parser, natural language processing (NLP) engine, analyzer, ontology engine, interface and integrator. It also includes a structural and an ontological database and a dictionary. It also includes a preliminary ontology crawler, which crawls the intranet for ontologies that it can merge and link to existing ontologies on the home server. The parser performs the preliminary tasks such as removing extraneous domain information and ensuring the given documents are in LOGS readable format and style. The NLP engine performs the tasks of morphological analysis and semantic analysis. It also performs the initial classification pass for identifying a root. The analyzer and ontology engine work closely in an iterative process. The interface module provides an entry point for the domain expert or an application programmer to alter the ontology generation process, such as in controlling various parameters and reordering discovered nodes. The integrator module aids in the deployment of generated ontologies and enriched web pages, as also the integration of local ontologies into site-wide ontologies and vice versa.

Refinement process

Internally, the ontological information is stored as a lattice, conforming to the N3 triples. A major step in the generation process is the estimation of the principal component or the root node. We consider the ontology to be a lattice. With each iteration, we discover concepts of a lower order, which we expect to figure lower in the ontology. Thus, the root is discovered in the zeroth pass (the NLP engine), the first order concepts in the first iteration and so on. Concept identification and analysis is based on the relevancy signature algorithm [Rilof] with the modification that we use concept association rather than syntactic elements, and use the relations of the concepts rather than sentential structure. A preliminary training corpus and a neural net are available with LOGS. The training information is guided by heuristics generated by trial and error iterative analysis of related texts to find known patterns. The ontology engine uses the training patterns to identify a root or central concept initially and build a hierarchy of discovered concepts around it. The hierarchy is ordered according to the training patterns on concept organization provided a priori. The analyzer module essentially tries to merge unrelated lattices and split ontologies at vertices that are articulation points, to ensure that each constituent lattice is rich and well connected. It can also reorder roots and child nodes so that, the best-connected node is the root after each iteration. The iterations continue until the ontology before and after the current analyzer iteration are similar and the predetermined depth of analysis is met. With each application, we need to re-train to that domain for best results. Within a general domain, we would not need to retrain the system. Thus, we can analyze tutorials on C++ and java with the same application.

Algorithm

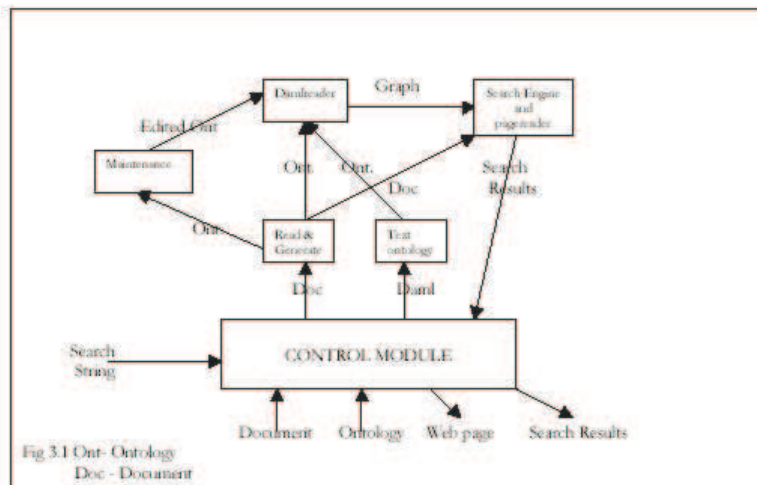
The following algorithm describes the process:

1. Identify root.
2. for I= 1 to orders
3. process texts
4. identify concepts in current level.
5. add all new concepts to lattice as new vertices
6. add all new relations found as edges
7. if any vertex is a free vertex, merge it to the main lattice. If a non-root element of any of the secondary lattices causes a connection to the main lattice, make it the root of the secondary lattice and merge it.
8. if any vertex has no peers, and has more than a threshold of edges coming into and leaving it, break the lattice there
9. after all breaks and merges, promote the vertex with the highest degree as the root in each lattice
10. next I
11. return complete ontology and secondary ontologies.

We term LOGS to be at once fast and lightweight as it is designed to process only small portions of the entire domain documents iteratively, to determine a conceptual model and relies on adherence to expected style and rules. It is lightweight because the training information for

the ontology engine, may not be comprehensive for all domains; indeed it is quite domain sensitive.

Sample Application:



To use LOGS in an end-to-end solution, we would need to extend the functionality of the NLP engine, supply training information as needed, and use the interface module to provide any domain expert feedback. Further, the ontology output from LOGS in the ontology database needs to be integrated with the end user interfaces such as web application or an intelligent tutorial system. The Eagle is an example of such an application tailored to programming language articles. The Eagle is a fairly comprehensive solution as it handles all tasks- detection of an ontology, maintenance and revision of ontology, retrieval, rendition and deployment. It is scalable, extensible and reconfigured with ease. At the top level, the Eagle does the following: from manually input definitive documents, the central concepts are identified and linked with proper role tags, into an ontology which can be ratified by a domain expert or a naïve user. The information core that is thus stored is used as aiding factors when a naïve user queries the system for information on a topic, for performing the searches. It is also used to generate the compendia, when possible. Compendia requests and queries are thus the input provided by the naïve users.

The user initially picks out the documents he wants to survey and serially serves them to the system. He can now plug in an ontology in the mark up language or in plain text or generate through the system. This information is resident and is not restricted to a single user session, whereas setting a search space is. A user run consists of one or more of the following interactions:

1. Creation of ontology
2. Modification of some ontology
3. Enhanced information search with context sensitivity

4. Summarize/learn etc.- this will generate a glossary like page for a topic. The use is to provide reference information, which may not be directly evident from the plain text source.
5. Detection: Finding relations between concepts from different parent documents.
6. Deployment. This is the main delivery vehicle, where a searchable web page is generated for the document and ontology specified.

Given a master document for study, different ontologies can be associated with it. For example, if we consider the Java overview page at <http://java.sun.com/docs/overviews/java/java-overview-1.html>, if an elaborate ontology were developed, the overall ontology would look like the figure 2, shown a few pages later, but a C++ programmer looking for what is new in java, and a systems administrator looking for its robustness and security would focus on entirely different areas of the ontology. This example can be easily extrapolated to huge ontobases, such as we see in the medical informatics area. Supposing they could tailor the ontology they want to suit their needs, their search would be more efficient. An ontology for this purpose could easily be thought of as a sublattice of the whole and therefore abstracted out of it for use.

So for greater context sensitivity, we allow the user to plug in the ontology of choice.

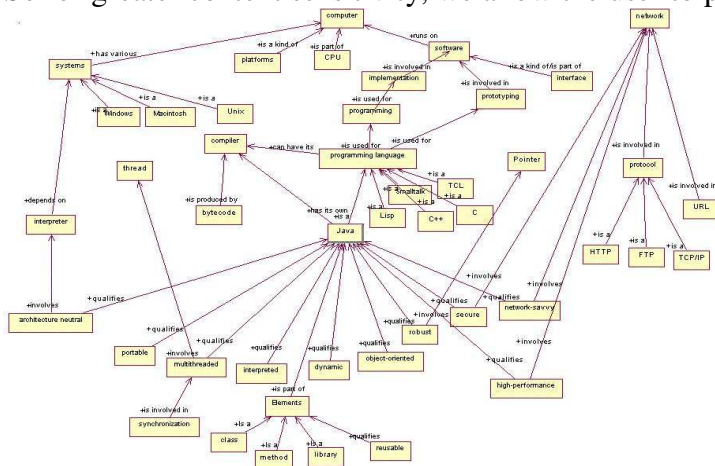


Figure 1 Java Ontology

The ontology generation and association steps are carried out before a set of documents are deployed. It is assumed that the user knows that the documents are related. Once the ontology is generated, several methods of exploitation and delivery are available. Using the tool to analyze a set of documents, generates an ontology and creates a set of webpages with embedded ontology information that can be deployed over a web server. This is the default, and context sensitive search algorithms will be tagged to the page. This enriched page could just be a DAML enriched web page. We could also generate a comprehensive intelligent tutorial system supported as an applet. While using the tool as an ontological server application, the daml enriched page served from the web server, would allow intelligent searches as also transactions with similar servers to augment the ontology. In the tutorial version, the ontology is tied to each page, for the external user, until there is redeployment. Another form of delivery is the context sensitive compendium, which produces a summary of

the requested and all related concepts. Finally, a primitive form of intelligent querying is also supported in the applet page. Every method of delivery also includes a feedback collection mechanism which automatically revises the ontology stored, by triggering the analyzer module of LOGS.

LOGS has been tested in 6 broad domains including news articles, programming languages and geography with appreciable results. Eagle successfully deployed interfaces for several programming language articles.

Future Work and Extending LOGS

LOGS is highly scalable, as the efficiency is based only on the information available to the training phase of the ontology engine. It is also very robust. To be able to use LOGS in another domain, we only need to modify the inputs to the ontology engine and the NLP engine. This too, could be from control corpuses for the new domain. For example, for Eagle, several java tutorial trails from the Sun Inc. website were used. Thus, although LOGS has been tested in intranets with a few web servers and relatively small amounts of ontological database content, to be able to work as part of a Semantic web application, substantial work on LOGS, on the import, export, revision and exploitation of ontologies from all over the internet, would be needed in future.

References:

Sintek, Abecker et al: *Using Ontologies for Advanced Information Access*. PAKeM 2000, The 3rd Int. Conference and Exhibition on The Practical Application of Knowledge Management, Manchester, UK.

Maedche, A. & Staab, S. *Mining Ontologies from Text*. In: EKAW-2000 - 12 th International Conference on Knowledge Engineering and Knowledge Management.

Abasalo A and Gomez G, *An Ontology Based Agent for Information Retrieval in Medicine*.

Sowa, John F., *Automating Ontology Development, article at the URL:*
<http://www.jfsowa.com/pubs/autotalk.htm>

Booz Allen and Hamilton DAML site
<http://www.davincinetbook.com:8080/daml/servlets/frame.html>

Kietz, Alexander Maedche, and Raphael Volz: *A Method for Semi-Automatic Ontology Acquisition from a Corporate Intranet*, In: Proc. of the EKAW'2000 Workshop "Ontologies and Texts".

OIL page at www.ontoknowledge.org

DAML+OIL page at W3C.org and DAML.org

Pertarch Intelligent Web Search Agent White paper at www.mygolddigger.com

Aitken, S. Evaluation of an ontology-based information retrieval tool'
www.aiai.ed.ac.uk/~stuart/Papers/ontologyeval.ps

Shoe homepage at
www.cs.umd.edu/projects/plus/shoe

Gruber, T, web article : <http://www-ksl.stanford.edu/kst/>

Fensel, D. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, Berlin, 2001.

van Harmelen, F. & Fensel, D. Practical Knowledge Representation for the Web. *Proceedings of the IJCAI'99 Workshop on Intelligent Information Integration*, 1999.

FZI Ontology Development and Application Guide.

Zhu, X., et al. *Ontology-Based Web Site Mapping for Information Exploration*, 1999

Riloff, E. and Lorenzen, J., (1999) "Extraction-based Text Categorization: Generating Domain-specific Role Relationships Automatically" ([postscript](#), [pdf](#)) In *Natural Language Information Retrieval*, Tomek Strzalkowski, ed., Kluwer Academic Publishers

