A Web Browser Extension for growing-up Ontological Knowledge from Traditional Web Content

Maria Teresa Pazienza[a], Marco Pennacchiotti[b], Armando Stellato[a]

a) AI Research Group, DISP
University of Rome, Tor Vergata
{pazienza, stellato}@info.uniroma2.it

b) Computational Linguistics
Saarland University, Germany.
pennacchiotti@coli.uni-sb.de

Abstract

While the Web is facing interesting new changes in the way users access, interact and even participate to its growth, the most traditional applications dedicated to its fruition: web browsers, are not responding with the same euphoric boost for innovation, mostly relying on third party or open-source community-driven extensions for addressing the new Social and Semantic Web trends and technologies. This technological – and decisional – gap, which is probably due to the lack of a strong standardization commitment on the one side (Web 2.0/Social Web) and in the delay of massive adherence to new officially approved standards (W3C approved Semantic Web languages), has to be filled by successful stories which could lay the path for the evolution of browsers.

In this work we present a novel web browser extension which combines several features coming from the worlds of terminology and information extraction, semantic annotation and knowledge management, to support users in the process of both keeping track of interesting information they find on the web, and organizing its associated content following knowledge representation standards offered by the Semantic Web.

1. Introduction

In recent years several actions have been undertaken for supporting the end user in experiencing Semantic Web information through dedicated browsing facilities. This kind of "experience" ranges from browsing and editing RDF data through user-friendly interaction modalities (Fallenstein, 2004; Sauermann, 2005), as well as being able to annotate traditional web content with references to available and browsable ontologies; example of such tools are the Haystack web client (Quan & Karger, May, 2004), Magpie (Dzbor, Domingue, & Motta, 2003), Piggy-Bank (Huynh, Mazzocchi, & Karger, November, 2005) and others. Yet, what is still missing is a really integrated environment extending a web browser with (light) knowledge management facilities and efficient annotation and retrieval of acquired information, all in the hands of the user. Moreover, the great success of *light* web browser plug-ins (e.g. the Google^{TM} Toolbar, Del.icio.us bookmarking extension etc..), if compared to the scarce popularity of browsing solutions based on ad-hoc working environments like Eclipse, demonstrate that users prefer to "expand" their web experience still relying on their personal, traditional, web browser and try out new features which are not too intrusive for their usual way of working. A key requirement today is thus to keep such tools as simple, intuitive and efficient as possible, while preserving all the rich functionalities required by the Semantic Web.

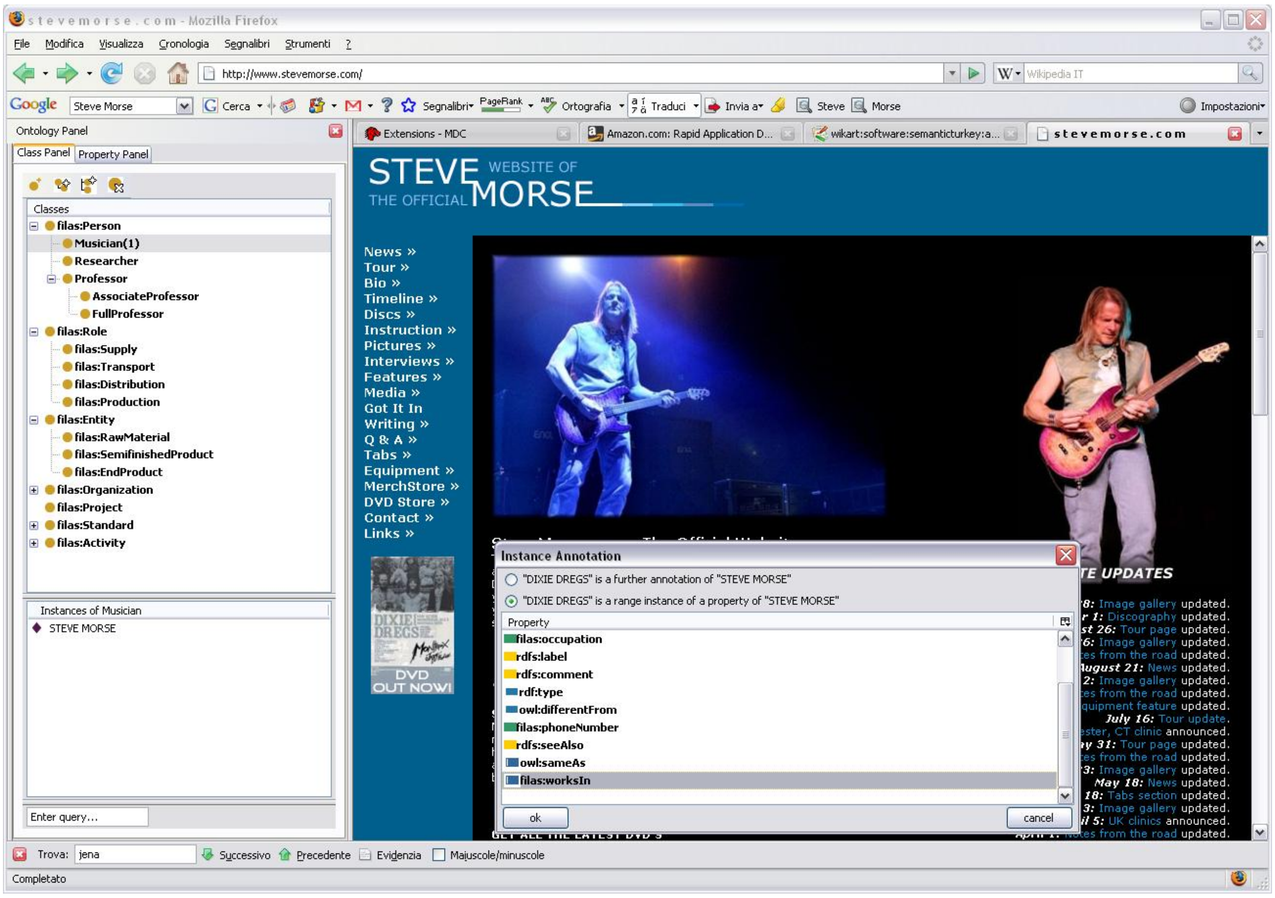
With such an objective, we have designed – and we are currently deploying – a platform for ontology editing and automatic web-based ontology population and development. Though beneficiating from past lessons, the platform offers a novel and unique combination of ontology editing and semantic annotation functionalities. Users can develop and edit structured semantic information, like in traditional ontology editing tools (Gennari, et al., 2003; TopBraid Composer), while at the same time be able to match it with associated textual references on traditional web documents. This last activity is not limited to semantic annotation, which is usually associated with mere annotation of textual data towards a reference ontology vocabulary, often bounded to a class hierarchy. Here, we aim at providing efficient user interaction modalities for building ontological data (from simple object creation to the instantiation of attributive and relational properties) while the user is naturally exploring and keeping track of web content. Our solution tries to get the best from both worlds (semantic markup and knowledge management) and combine their aspects in a unique approach. Another important aspect is the ability to automatically identify, extract and present relevant information in a manner that it can easily fit into available knowledge patterns inside the ontology that is being adopted/edited by the user.

The paper is organized as follows. In Section 2 we outline the general approach and design of the platform. In Section 3 we focus on the platform architecture and on its main functionalities. In Section 4, we present and evaluate its specific modules for ontology learning from texts. Finally, in Section 5, we draw final conclusions and outline future works.

2. Approach and Design

Our focus is to find innovative solutions for collecting, managing and retrieving data emerging as relevant while surfing the web; then, structuring them by following personal criteria. In such view, the main functionalities to be provided to the end user relate to:

- **Ontology editing**. Creation of an ontology from scratch by the user, with possibility of importing other ones from file system, web, etc..
- **Ontology learning from text**. Semi-automatic creation of an ontology from text, acquisition of new concepts and relations from text by using Natural Language Processing (NLP) techniques.
- **Ontology population**. Drag and drop of instances as well as their different lexical occurrences from


stevemorse.com - Mozilla Firefox
File Modifica Visualizza Cronologia Segnalibri Strumenti ?
http://www.stevemorse.com/
Ontology Panel
Class Panel Property Panel
Classes
filas:Person
Musician(1)
Researcher
Professor
AssociateProfessor
FullProfessor
filas:Role
filas:Supply
filas:Transport
filas:Distribution
filas:Production
filas:Entity
filas:RawMaterial
filas:SemifinishedProduct
filas:EndProduct
filas:Organization
filas:Project
filas:Standard
filas:Activity
Instances of Musician
STEVE MORSE
Enter query...
STEVE MORSE THE OFFICIAL WEBSITE OF
News » Tour » Bio » Timeline » Discs » Instruction » Pictures » Interviews » Features » Media » Got It In Writing » Q & A » Tabs » Equipment » MerchStore » DVD Store » Contact » Links »
DIXIE DREGS DVD OUT NOW!
Trova: jena Successivo Precedente Evidenzia Majuscole/minuscole
Completato

Instance Annotation

○ "DIXIE DREGS" is a further annotation of "STEVE MORSE"

● "DIXIE DREGS" is a range instance of a property of "STEVE MORSE"

Property
filas:occupation
rdfs:label
rdfs:comment
rdf:type
owl:differentFrom
filas:phoneNumber
rdfs:seeAlso
owl:sameAs
filas:worksIn

ok cancel

Figure 1: semantic annotation and ontology building combined in a few mouse clicks

web pages and texts or by use of automatic techniques.

To fulfill these objectives, we have brought new ideas and functionalities inside our Semantic Web platform Semantic Turkey (Griesi, Pazienza, & Stellato, 2007). Semantic Turkey, an extension for the popular web browser Mozilla Firefox, can be seen (and has been originally conceived) as a personal desktop solution for organizing and managing – inside RDF/OWL ontologies – the relevant information caught during web navigation, thus replacing past ordinary solutions like bookmarks. The original intent of Semantic Turkey was to bring innovative aspects borrowed from new Semantic Web technologies into the everyday life of the typical web user: its main paradigmatic innovation, with respect to traditional bookmarking, resides in the fact that it defines a clear separation between knowledge data (the WHAT) and web links (the WHERE), so that the user can focus on organizing the collected information according to his preferences while keeping it, on a complete different perspective, updated with pointers to the web resources where it is referenced.

The requirements and design goals which have been satisfied in the first prototype of Semantic Turkey are:

1. Capturing information from web pages, both by considering each page as a whole, as well as by annotating portions of its text
2. Editing of a personal ontology for categorizing annotated information and, possibly, for exchanging data with other users. Importing and combining other existing ontologies into one domain space is possible as well.
3. Navigation of the structured information as an underlying semantic net through the links to the web sources where it has been annotated.
4. Clear separation between business model and user interface, by adopting a "knowledge service" architecture.

While the main stream of effort was to create a robust, usable platform for semantic bookmarking, in this version we tried to boost the intelligence of the system, by allowing for automatic extraction of information from semistructured (web pages in general) and structured data (like tables) which still has no explicit semantics associated to its content, to populate the working ontologies with new instances (both in terms of domain objects and relationships) or even increment its domain theory with new concepts and relations.

3. Semantic Turkey Architecture

The architecture of Semantic Turkey consists of a web application, designed using a three layered approach.

3.1. Presentation Layer

This layer has been developed as an extension for the web browser Firefox (Firefox web browser | Faster, more secure, & customizable). This approach has two main advantages:

- total reuse of the functionalities of a well assessed, stable and complete software for web browsing,
- a non intrusive offer for the user, who can still adopt the web browser he has been acquainted with.

3.2. Services Layer

It has been realized through a collection of Java Web Services, published through the embedded Web Server "Jetty" (Jetty Java HTTP Servlet Server).

Jetty is implemented entirely in Java, and the architecture foresees its use as an embedded component. This means that the Web Server and the Web Application run in the same process, without interconnection overheads and other sort of complications.

This solution also allows for a flexible use of the tool, since it can both be adopted as a completely autonomous web browser extension, as well as a personal access point for collaborative web exploration and annotation.

3.3. Persistence Layer

The persistence layer includes the component for managing the ontology, which is represented in the OWL language (Web Ontology Language webpage). This layer has been developed by using the Sesame ontology library (Broekstra, Kampman, & van Harmelen, 2002) and its OWLIM plugin (Kiryakov, Ognyanov, & Manov, 2005). Sesame is an open source RDF database with support for RDF Schema inference and querying. For what concerns the knowledge model of Semantic Turkey, there are two different layers of ontological knowledge: Application Layer and User Layer.

The *Application Layer* contains ontologies required by the application for coordinating and organizing its services. These ontologies are hidden by default to the user.

In the core version of Semantic Turkey, this layer includes the sole Semantic Annotation ontology, which provides concepts and relations for keeping track of user semantic bookmarks.

The User layer includes all the knowledge domain which is handled by the user: data imported from the web, personally defined data as well as information annotated from web pages.

3.4. Semantic Navigation

Semantic Navigation option can be accessed as an additional feature, letting the user graphically explore the ontology. A Java applet will be loaded on a new tab of the browser displaying the graph view of the ontology, allowing the user to navigate its content and get back to the pages related to the annotated knowledge.

Conversely, Semantic Turkey reports to the user, through a dedicated status bar, the pages which have been previously annotated.

4. Learning ontological knowledge from texts

Automatic methods for extracting knowledge from texts are today mature enough to be integrated and leveraged in real NLP applications, such as our platform. These methods include techniques for extracting terminologies (Pazienza, Pennacchiotti, & Zanzotto, 2005), concept lists (Lin & Pantel, 2002), relations among terms (Pantel & Pennacchiotti, 2006), and others.

Ontology learning from text (Buitelaar & Cimiano, 2008) may be considered as a means to (semi-) automatically build ontologies from document collections, by using unsupervised techniques supported by a final human validation over the extracted data. As a matter of fact, in recent years many tools have been created for semi-automatically supporting human experts in the task of ontology building from documents, such as KIM (Popov, Kiryakov, Kirilov, Manov, Ognyanoff, & Goranov, 2003) Text-to-Onto (Mädche, 2000) and more.

Yet, even if semi-supervised extraction methods guarantee a good level of accuracy, they often entail high computational and time costs. In facts, they generally involve deep syntactic parsing, statistical computations, and the exploration of large hierarchies and lexical knowledge bases (e.g. WordNet), which require time and processing resources. In most cases, these costs prevent a successful integration of extraction techniques into ontology-related platforms.

Here, our major challenge is then to integrate modules for ontology learning into Semantic Turkey, keeping them efficient and fast, while preserving a good level of accuracy. To do that, we both focus on the optimization of existing and new techniques, and on a careful use of shallow NLP tools for text analysis (e.g. lemmatizers, fast PoS-taggers) leaving aside deeper and more costly tools (e.g. full syntactic and semantic parsers). By following this approach, and to support the end user in all steps of the learning process, we are integrating in the platform the following NLP-based modules::

- ***Terminology Extractor:*** a module to extract and validate terminological expression in web pages;
- ***Relation Extractor:*** a module to harvest binary semantic relations from web pages and upload them into the ontology;
- ***Table Knowledge Extractor:*** a module to automatically extract ontological information (classes, instances and properties) from structured data.

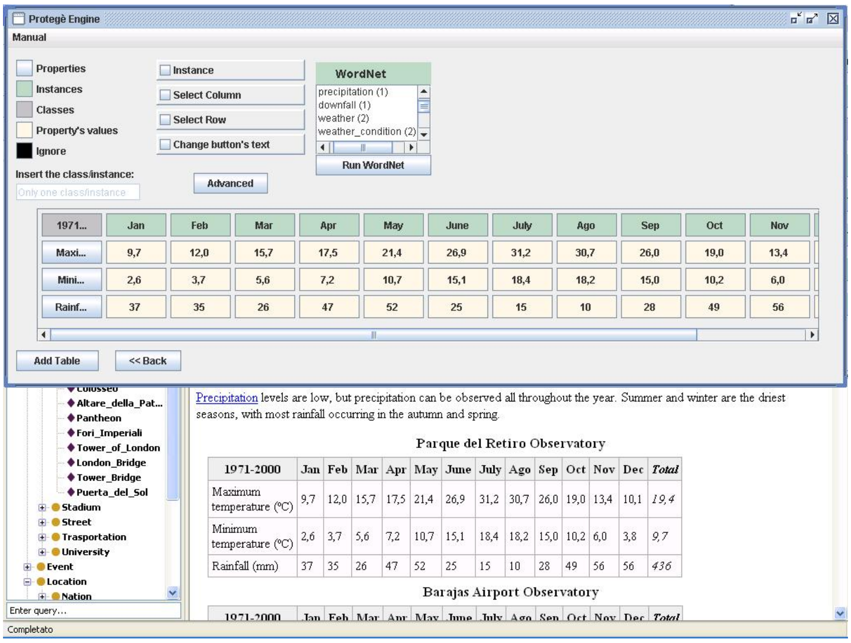
In the rest of this section we will describe in detail the above modules.

4.1. Terminology Extractor

A term is commonly defined as "a surface representation of a specific domain concept" (Jacquemin, 1997). Terms can be then considered as candidate concept instances for an ontology and as fundamental pieces in ontology learning. Our terminology extractor is implemented by using shallow techniques, based on regular expression recognition of candidate terms over PoS-tagged data. This guarantees a high level of accuracy, while preserving computational efficiency. We use well-assessed regular expressions such as (Justeson & Katz, 1995):

((Adj|Noun)+|((Adj|Noun)*(NounPrep)?)(Adj|Noun)*)Noun

which allow to capture both simple terms (e.g. "underground economy") and complex ones (e.g. "Iraqi National Joint Action Committee for Reforms"). Once candidates are extracted, we apply statistical measures (e.g. frequency, pmi) to select the most reliable terms, and we propose them to the user for final validation



1971...	Jan	Feb	Mar	Apr	May	June	July	Ago	Sep	Oct	Nov
Maxi...	9,7	12,0	15,7	17,5	21,4	26,9	31,2	30,7	26,0	19,0	13,4
Mini...	2,6	3,7	5,6	7,2	10,7	15,1	18,4	18,2	15,0	10,2	6,0
Rainf...	37	35	26	47	52	25	15	10	28	49	56

Precipitation levels are low, but precipitation can be observed all throughout the year. Summer and winter are the driest seasons, with most rainfall occurring in the autumn and spring.

Parque del Retiro Observatory

1971-2000	Jan	Feb	Mar	Apr	May	June	July	Ago	Sep	Oct	Nov	Dec	*Total*
Maximum temperature (ºC)	9,7	12,0	15,7	17,5	21,4	26,9	31,2	30,7	26,0	19,0	13,4	10,1	*19,4*
Minimum temperature (ºC)	2,6	3,7	5,6	7,2	10,7	15,1	18,4	18,2	15,0	10,2	6,0	3,8	*9,7*
Rainfall (mm)	37	35	26	47	52	25	15	10	28	49	56	56	*436*

Barajas Airport Observatory

1971-2000	Jan	Feb	Mar	Apr	May	June	July	Ago	Sep	Oct	Nov	Dec	*Total*

Figure 2: validation interface of the *table knowledge extractor* module.

4.2. Relation Extractor

Relation extraction is intended as the task of extracting generic and specific binary semantic relations between terms from a corpus, such as *is-a(bachelor, man)* and *capital-of(Roma, Italy)*. This can be seen as a second step in ontology learning, where relations among concept instance are discovered. Our module implements a pattern-based technique similar to (Hearst, 1992) and to the state of the art (Pantel & Pennacchiotti, 2006), but applies specific strategies to keep the algorithm efficient, such as the use of simplified statistical measure and of shallow analyzers. The module guarantees: (1) *minimal supervision*, by using as input one of few instance(s) already in the ontology, and by presenting as output a list of ranked instances which can be easily validated and uploaded; (2) *high accuracy*, by adopting a dedicated reliability measure to weight the extracted instances; (3) *easing data sparseness*: by implementing specific techniques to exploit long distance dependencies; (4) *generality*, as it is applicable to a wide variety of relations. Hereafter we present the different components of the module.

Input Interface. This provides to the user the functionalities needed for starting the relation extraction process, by allowing to select a seed pair $s=(x_s,y_s)$ for a given relation. The pair consists in two entities of the ontology, related by an object property. The extraction process is then executed as follows.

Pattern Induction and Expansion. Given an input seed instance *s*, the algorithm looks in the corpus for all sentences containing the two terms. These sentences are parsed by the *Chaos* constituent-dependency parser (Basili & Zanzotto, 2002). All dependency paths connecting the seed words are extracted as patterns *P*. The use of Chaos guarantees two main advantages with respect to simple surface approaches such as (Ravichandran & Hovy, 2002). First, the use of dependency information allows to extract more interesting patterns. Second, Chaos explicitly represents ambiguous relations between constituents, allowing to infer patterns also when the syntactic interpretation is not complete. Figure 7 shows a parsed sentence connecting the seed *s=(Madrid,Spain)*. The algorithm extracts as patterns the paths: *"X is the capital of Y"* and *"X is of Y"*, which connect the two words. Notice that a simple surface approach would have extracted the only irrelevant pattern *"X since 1561 is the capital of Y"*. The dependency analysis allows to extract more useful patterns, helping to deal with data sparseness. Yet, the algorithm is prone to capture too generic patterns such as *"X of Y"*.A reliability measure is applied to cope with this problem. Also, to further deal with data sparseness, the algorithm expands the patterns *P* in a bigger set *P'*, by including different morphological variations of the main verb (e.g. "*X being the capital of Y*", *"X was the capital of Y"*, etc.).

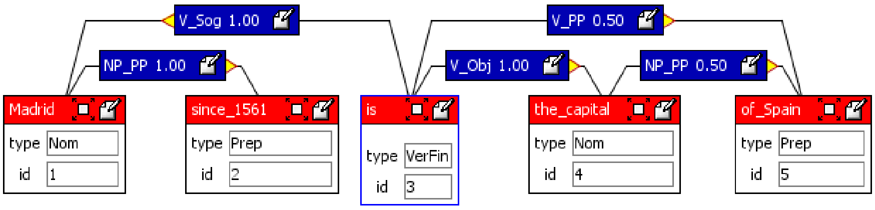


Figure 3: A dependency graph output by *Chaos*. *Lower boxes* are constituents. *Upper boxes* are dependencies, with the related *plausibility*, i.e. a representation of ambiguous relations.

Instance Induction and Reliability Ranking. Given the set *P'*, the algorithm retrieves all sentences containing the words of any *p∈P'*. Each sentence is then parsed by *Chaos*. The constituents connected by a dependency path corresponding to a pattern in *P'* are extracted as new instances *I*. For example, the new instance *(New Delhi, India)* is extracted from: *"New Delhi being the capital of India, is an important financial market"*. Each instance *i=(x,y)∈I* is assigned a *reliability score R(i)*, accounting for the intuition that an instance is reliable, i.e. it is likely to be correct, if: *(1)* it is activated by many patterns; *(2)* the Part-of-Speech (PoS) of the instance and of the seed *s* are the same; *(3)* the semantic class of *x* and *y* are respectively similar to those of x_s and y_s.:

$$R(i) = \alpha \frac{|P_i|}{|i|} + \beta POS_i + \gamma\left(\frac{1}{k} + \frac{1}{j}\right)$$

where P_i are the patterns activating *i*; POS_i is a binary value which is 1 if the PoS of *i* and *s* are the same, 0 otherwise; *k* and *j* are the depths of the *least common subsumer* respectively between *x* and x_s and between *y* and y_s in the WordNet hyperonymy hierarchy. *α, β,* and *γ* parameters sum to 1, weighting the contribution of respectively point *(1)*,*(2)* and *(3)*.

Validation Interface and Ontology Uploading. The ranked list of extracted instances *I* is presented to the user, via a validation interface, which allows the user to select the instances to be uploaded in the ontology. Once validation is finished,: *x* and *y* of each instance *(x,y)* are inserted in the same ontology class of x_s and y_s and the related object property is activated. For example, if x_s*=Madrid* is an instance of the ontology class *city* and y_s*=Spain* is instance of *nation*, the new ontological entities *New Delhi* and *India* are added as instances of the class *city* and *nation*, and related by the object property *capital-of.*

We measured the performance of the relation extraction algorithm on the task of extracting *capital-of* and *located-in* relations instances, over a domain corpus of 80 Wikipedia pages on European and Asian cities (207,555 tokens). We set the seed instance to *s=(Madrid,Spain)*, and the parameters *α, β,* and *γ* to 0.05, 0.25 and 0.75, by estimation on a small annotated development corpus of 10 pages. As gold standard reference we used a list of instances I_{gs} manually extracted from the corpus. We measure performance in term of *precision P*, *relative-recall R*, *F-measure*, and *goldStd-recall G* at different levels of a threshold *τ*. The set of instances $I_\tau \in I$ which have a score *R(i)* above the threshold are taken as accepted by the system. At each level of *τ*, P_τ and R_τ , F_τ and G_τ are defined as follows:

$$P_\tau = \frac{|I_\tau \cap I_{gs}|}{|I_\tau|} \quad R_\tau = \frac{|I_\tau \cap I_{gs}|}{|I|} \quad F_\tau = 2\frac{P_\tau * R_\tau}{P_\tau + R_\tau} \quad G_\tau = \frac{|I_\tau \cap I_{gs}|}{|I_{gs}|}$$

GoldStd-recall is intended to capture the recall over the gold standard, while relative-recall captures recall at a given threshold over all extracted instances.

Results for the *capital-of* relation are reported in Figure ??. In all, the algorithm extracted around 50 instances for both relations. In general our algorithm is able to extract instances with high precision and recall. For example, at *τ*=0.5, precision is high (almost 0.90) while goldStd recall is still acceptable, about 0.45. Precision is up to that obtained by state of the art algorithms: for example (Pantel & Pennacchiotti, 2006) obtain 0.91 on a chemistry corpus of the same size as our for the *reaction* relation. Yet, our recall is lower, as we do not exploit generic patterns. Figures indicates that according to the intuition, as the threshold grows, precision improves, while recall decreases, indicating that our reliability measure is coherent and can be effectively used to select correct/incorrect instances. From a qualitative perspective, most of the erroneous extracted instances correspond to parsing errors or to the induction of wrong patterns (e.g. the incorrect instance *(Antananarivo, University)* for the *capital-of* relation is fired by the wrong pattern *"X is home of Y"*).

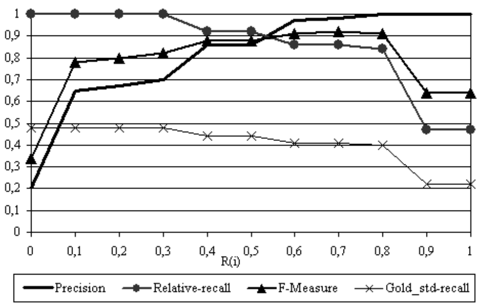


Figure 4: Precision, Relative-Recall, F-measure and GoldStd-Recall at different levels of *τ* for the relation *capital-of.*

4.3. Table Knowledge Extractor

Structured information, such as tables and lists, offer a rich source of knowledge to enrich ontologies, as they have the major advantage that the data they contain are coherently organized, making their ontological interpretation easier with respect to unstructured texts. Also, they typically contain *dense* meaningful content which tends to be ontology-oriented. Despite this, not much attention has been paid so far on the extraction of ontological information from tables, exception being the TARTAR (Pivk, Cimiano, Sure, Gams, Rajkovič, & Studer, 2007) and the TANGO (Tiberino, Embley, Lonsdale, Ding, & Nagy, 2005) systems, which unfortunately are not intended to be integrated in an ontology editing architecture, as we would.

Our module aims at extracting knowledge (namely, classes, instances and properties) from tables in HTML pages, and then propose a complete ready-for-validation ontological interpretation to the user.

In our framework, a table is intended as a matrix of cells (see Figure 5), whose structure can be divided in four main areas, which in most cases contain different type of information: (1) *First row* (cells *<1,2> ... <1,n>*), which usually contains a *column header*, i.e. a short description of the information enclosed in each column. (2) *First column* (cells *<2,1> ... <2,m>*), typically containing a

<1,1>	<1,2>	<1,3>	…	<1,n>
<2,1>	<2,2>	<2,3>	…	<2,n>
…	…	…	…	…
<m,1>	<m,2>	<m,3>	…	<m,n>

Figure 5: Structure of a *n*x*m* table

row header, describing the content of a single row. (3) *First cell* (cell *<1,1>*), sometimes used to give a short indication on the type of data contained in the table (*table header*); in other cases, it is part of the first row or the first column. (4) *Internal cells* (other cells), containing the actual *data* of the table, whose meaning is described by the related cells in the first row/column.[1] The structure indicates that from an information-content perspective, a table is either: *three-dimensional*, containing row header, column header and data; or *two-dimensional*, when either the row or column header is not present.[2] Also, we can assume two basic facts: *(a)* in most cases tables contain simple *flat* non-hierarchical knowledge; *(b)* a table cannot contain knowledge about more than one class, as this would imply the table to have a fourth dimension to represent class names. Following these assumptions we can ontologically classify tables in three categories:

- *Class Tables:* containing a class definition and a set of its instances. The information that has to be enclosed in the table are: property names, instance names and property values. The table must then be three-dimensional, i.e. *it must have row header and column header*. Property names and instance names can be either reported in the row or column headers. Property values are reported in the internal cells, while the class name, if present, is typically in the table header.
- *Instance Tables:* containing information about a single instance. The information enclosed in the table are property name and property value. *These tables are typically two dimensional and have either exactly two columns or rows*. The column (row) header indicates the name of the properties, while the second column contains the property values..
- *Empty tables:* these are tables which do not contain any ontological interesting content (e.g.graphical elements).

Our module leverages the above classification to extract knowledge performing the following steps.

Table extraction and selection. Given an input Web page displayed in the browser, the module. extracts all well-formed tables. In the case of nested tables, the outermost is retained as most informative (inner tables tend to contain in most cases graphical objects). The user can then selects in the graphical interface, the set of tables to analyse.

Table type identification. Given an input table, the module applies a cascade of heuristics to classify it either as *class table*, if it is three-dimensional, or as *instance table*, if it is two-dimensional. The heuristics do so by analyzing the number of columns, and by checking the presence of column and row headers considering textual and stylistic properties. For example, if the first row and column have a background colour different from the other cells, they are respectively identified as column and row header; then, the table is classified as three-dimensional – i.e. as class table.

Ontological analysis. Given a classified table, the module infers the contained ontological entities. In the case of *class tables* entities are: properties' names of the class, instances' names and related property values. Instance and property names can be alternatively coded in the row or in the column headers, while the property values are in the internal cells. The issue is then to understand which header contains which names. For this purpose, the assumption is that an ontological property has always the same range: if the column header contains the property names, the first element of a column (property name) must be followed by cells of the same data type (property values). The same observation stands for the row header. If all internal cells are of the same data type, the system simply guesses as default that the column header represents the property names. In the case of *instance tables* the analysis is straightforward: the left column (row header) contains property names, and the right column containing the properties' values (internal cells). At the end of the ontological analysis, each cell in the tables is assigned an ontological type.

Validation and Ontology Uploading. The results of the ontological analysis are shown to the user (see Figure 2), that can then decide either to reject the table as not interesting, to accept completely the interpretation, or to modify it. In the latter case, the user is provided with different tools to change the ontological type of cells. Once the correct ontological interpretation of the table is decided, the information is automatically uploaded in the ontology. In this last phase the user has to specify the class/instance name, and the ontological attachment (the parent class for class tables, the referring class for instance table).

We verified the performance of the relation extraction module on a corpus of 100 Web pages of European and Asian capitals from *Wikipedia*, amounting to 207 tables. We computed the accuracy on *table type identification* (i.e classification in *class* or *instance table*), and the accuracy on *ontological interpretation* (i.e. predicting a completely correct interpretation of the table, in all cells). We obtained respectively accuracy of 0.91 and 0.77. Results show that the systems' heuristics are very accurate in predicting the correct table type, and highly reliable on the ontological interpretation, revealing that the simple heuristics implemented are effective.

5. Conclusions and future work

In this paper we presented a novel architecture for collaborative ontology editing, which offers a unique combination of ontology editing and semantic annotation functionalities, and focusing on the use of "light" NLP techniques for semi-automatically supporting the ontology building process. We described three of these NLP modules, which extract ontological information from Web texts, guaranteeing an high level of accuracy.

In the future, we plan to make available the architecture and its NLP modules to the community. We are also planning to integrate new NLP engines, to extract other type of ontological knowledge, such us events and situational frames.

[1] Though this typical structure is verified in most cases, our module is able to detect and treat exceptions.

[2] Mono-dimensional tables are seldom, as the value of internal rows and columns must be somehow described by a header.

6. References

Basili, R., & Zanzotto, F. (2002). Parsing engineering and empirical robustness. *Natural Language Engineering 8/2-3* , 1245-1262.

Broekstra, J., Kampman, A., & van Harmelen, F. (2002). Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. *The Semantic Web - ISWC 2002: First International Semantic Web Conference* (p. 54-68). Sardinia, Italy: Springer Berlin / Heidelberg.

Buitelaar, P., & Cimiano, P. (2008). *Ontology Learning and Population: Bridging the Gap between Text and Knowledge.* IOS Press.

Dzbor, M., Domingue, J., & Motta, E. (2003). Magpie: Towards a Semantic Web Browser. *2nd International Semantic Web Conference (ISWC03).* Florida, USA.

Fallenstein, B. (2004). Fentwine: A navigational RDF browser and editor. *1st Workshop on Friend of a Friend, Social Networking and the Semantic Web (FOAF).* Galway.

Firefox web browser | Faster, more secure, & customizable. (n.d.). Retrieved from http://www.mozilla.com/en-US/firefox/

Gennari, J., Musen, M., Fergerson, R., Grosso, W., Crubézy, M., Eriksson, H., et al. (2003). The evolution of Protégé-2000: An environment for knowledge-based systems development,. *International Journal of Human-Computer Studies , 58* (1), 89–123.

Griesi, D., Pazienza, M. T., & Stellato, A. (2007). Semantic Turkey - a Semantic Bookmarking tool (System Description). *4th European Semantic Web Conference (ESWC 2007).* Innsbruck, Austria.

Hearst, M. (1992). Automatic acquisition of hyponyms from large text corpora. *Proceedings of COLING-92*, (pp. 539-545). Nantes, France.

Huynh, D., Mazzocchi, S., & Karger, D. (November, 2005). Piggy Bank: Experience the Semantic Web Inside Your Web Browser. *Fourth International Semantic Web Conference (ISWC05)*, (p. 413-430). Galway, Ireland.

Jacquemin, C. (1997). *Variation terminologique: Reconnaissance et acquisition automatiques de termes et de leurs variantes en corpus.* Universitè de Nantes,France: Mémoire d'Habilitation à Diriger des Recherches en informatique fondamentale.

Jetty Java HTTP Servlet Server. (n.d.). Retrieved from http://jetty.mortbay.org/jetty/

Justeson, J., & Katz, S. (1995). Technical Terminology: some linguistic properties and an algorithm for identification in text. *Natural Language Engineering, 1* , 9-27.

Kiryakov, A., Ognyanov, D., & Manov, D. (2005). OWLIM – a Pragmatic Semantic Repository for OWL. *Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2005), WISE 2005.* New York City, USA.

Lin, D., & Pantel, P. (2002). Concept discovery from text. *20th International Conference on Computational Linguistics (COLING-02)*, (p. 577–583). Taipei, Taiwan.

Mädche, A. (2000). The TextToOnto Ontology Learning Environment. *ICCS 2000.* Darmstadt: Vortrag.

Pantel, P., & Pennacchiotti, M. (2006). Espresso: A Bootstrapping Algorithm for Automatically Harvesting Semantic Relations. *COLING/ACL-06.* . Sydney, Australia.

Pazienza, M., Pennacchiotti, M., & Zanzotto, F. (2005). Terminology extraction: an analysis of linguistic and statistical approaches. In S. (Ed.), *Knowledge Mining, , Series: Studies in Fuzziness and Soft Computing,,* (p. Vol.185). Springer .

Pivk, A., Cimiano, P., Sure, Y., Gams, M., Rajkovič, V., & Studer, R. (2007). Transforming arbitrary tables into logical form with TARTAR. *Data & Knowledge Engineering, 60:3* .

Popov, B., Kiryakov, A., Kirilov, A., Manov, D., Ognyanoff, D., & Goranov, M. (2003). KIM – Semantic Annotation Platform. *2nd International Semantic Web Conference (ISWC2003). 2870*, p. 834-849. Florida, USA: Springer-Verlag Berlin Heidelberg.

Quan, D., & Karger, D. (May, 2004). How to Make a Semantic Web Browser. *Thirteenth International World Wide Web Conference (WWW2004).* New York City, USA.

Ravichandran, D., & Hovy, E. (2002). Learning surface text patterns for a question answering system. *Proceedings of ACL-2002*, (pp. 41-47). Philadelphia, PA.

Sauermann, L. (2005). The Gnowsis Semantic Desktop for Information Integration. *1st Workshop on Intelligent Office Appliances(IOA 2005): Knowledge-Appliances in the Office of the Future.* Kaiserslautern, Germany.

Tiberino, A., Embley, D., Lonsdale, D., Ding, Y., & Nagy, G. (2005). Towards Ontology Generation from Tables. *World Wide Web: Internet and Web Information Systems:8* , 261-285.

TopBraid Composer. (n.d.). Retrieved from http://topbraidcomposer.info/

Web Ontology Language webpage. (n.d.). Retrieved from W3C: http://www.w3.org/TR/owl-features/