T-REX: A Domain-Independent System for Automated Cultural Information Extraction

Massimiliano Albanese
Univ. of Maryland Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742
albanese@umiacs.umd.edu

V.S.Subrahmanian
Computer Science Dept. and UMIACS
University of Maryland
College Park, Maryland 20742
vs@cs.umd.edu

Abstract

RDF (Resource Description Framework) is a web standard defined by the World Wide Web Consortium. In RDF, we can define *schemas* of interest. For example, we can define a schema about tribes on the Pakistan-Afghanistan borderland, or a schema about violent events. An RDF *instance* is a set of facts that are compatible with the schema. The principal contribution of this paper is the development of a scalable system called T-REX (short for "The RDF EXtractor") that allows us to extract instances associated with a user-specified schema, independently of the domain about which we wish to extract data. Using T-REX, we have successfully extracted information about various aspects of about 20 tribes living in the Pakistan-Afghanistan border. Moreover, we have used T-REX to successfully extract occurrences of violent events from a set of 80 news sites in approximately 50 countries. T-REX scales well – it has processed approximately 45,000 web pages per day for the last 6 months.

Introduction

There is a huge shortage in the US about detailed information about diverse cultural groups in other parts of the world. For example, US forces in Afghanistan are continuously learning about the tribes on the Pakistan-Afghanistan border. Given one of these tribes (e.g. the Afridi), they would like to learn the answers to questions such as: *What is the source of economic support for the Afridi tribe? Which other tribes have they had conflicts with, and over what issues did these conflicts arise? Have they been involved in violence against the US? Against other tribes in the region?* Alternatively, if we wish to have a real-time "violence watch" around the world, we may need to define what constitutes a violent event, and what types of attributes about a violent event are of interest. For instance, we may wish to identify the *victims, number of dead, number of injured, perpetrators, location, time, instrument used* and other attributes of a given violent event.

In this paper, we provide a brief overview of the T-REX system. T-REX is a generic, domain-independent system which takes a schema as input. Informally speaking, a schema specifies the types of information we want to extract. T-REX does not care whether the schema involves tracking health care phenomena or information about businesses or information about tribes or information about violent events. The schema can be about anything. When we wish to specify a schema about violent events, we merely need to specify attributes such as those mentioned in the preceding paragraph. The T-REX system currently searches through 80 online news sites from 50 countries around the world. Everyday, it examines about 45,000 articles and tries to extract instances of a schema from those articles.

The outline of this paper is as follows. Following section describes the overall software architecture of T-REX. Then we give the specifications of our multilingual annotation interface – this component is used to create an annotated corpus of sentences that are the basis to learn extraction rules. The technique used to learn such extraction rules is described in the subsequent section. We then describe the algorithm to extract annotated RDF triples from news articles using the corpus of rules. The extracted triples are stored in a specialized *annotated RDF database* that we have separately designed (Udrea, Recupero, & Subrahmanian 2006). In the last two sections of the paper we give some details of the implementation and discuss related works respectively.

T-REX Architecture

Figure 1 shows the prototype architecture of our T-REX system. T-REX is part of a broader architecture called CARA (Cultural Reasoning Architecture) that provides a framework for reasoning about how diverse cultural, political, industrial, and other organizations make decisions (Subrahmanian *et al.* 2007).

The T-REX system architecture consists of several components. The system works through a Multilingual Annotation INterface (MAIN). MAIN presents human annotators with sentences, one at a time. For each sentence, MAIN presents a parse tree and asks the annotator to add meta information to the nodes in the parse tree and to specify relationships between the nodes.

A Rule Extraction Engine takes each annotated sentence as input and tries to learn an "extraction rule" from it. These extraction rules tell the system how to extract data from new sentences (or fragments of sentences) that are encountered when processing real time news feeds.

The set of rules learned in this manner is then applied to real time news feeds. The rules generate what are called

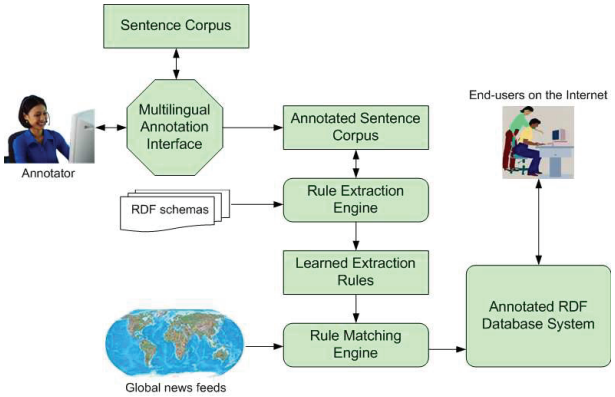


Figure 1: TREX architecture

annotated RDF triples (Udrea, Recupero, & Subrahmanian 2006). A triple in RDF (World Wide Web Consortium (W3C) 2004) has the form `(subject, property, object)`. For example, suppose we are looking at some violent event (say `KillingEvent8`). This event may have "Hazara Afghans" as the value of its `victims` property. Thus, `(KillingEvent8, victims, Hazara Afghans)` is an RDF triple generated in this way. Another triple might be `(KillingEvent8, location, Mazar-e-Sharif)` saying that this particular event occurred in Mazar-e-Sharif. Each triple can have an annotation. An example annotation is the set of news sources that validate a given triple.

Once the learned rules are used to generate triples, the triples are stored into an *Annotated RDF Database* system that we have separately designed (Udrea, Recupero, & Subrahmanian 2006). An end user can query this database in the usual ways supported by relational DBMSs.

Multilingual Annotation Interface

The *Multilingual Annotation INterface* (MAIN) is a web-based tool designed to allow human annotators to create an annotated corpus of sentences in different languages. The creation of an annotated corpus is a fundamental step to enable our system to effectively process news feeds and extract the desired information. Processing a news article and representing its information content in some structured format is not a trivial task for an automated system as it would be for a human: computers cannot understand natural language.

Consider as an example the sentence "*At least 73 civilians were killed February 1 in simultaneous suicide bombings at a Hilla market*". The sentence clearly reports a violent event in which someone was killed. If asked to identify the victims, the number of victims, the place and the time when the event took place, any human would easily reply that the victims were "civilians", the number of victims was "at least 73", the place was "a Hilla market" and the date was "February 1". Without appropriate models it would not be possible for an algorithm to understand that the sentence describes a "killing event" and identify all the attributes of such event.

The computational problem becomes even more complex if we notice that the same piece of information can be delivered by many slightly different variations of the above sentence. Just to list a few, consider the following two sentences that report the same event, but using a few different words or different levels of accuracy for dates and quantities.

- "*More than 73 civilians were massacred in February in suicide attacks at a Hilla marketplace*"
- "*74 people were killed on February 1, 2007 in multiple bombings at a Hilla market*"

We also notice that other similar events may be reported through similar sentences, describing the same set of attributes as in the previous case. As an example, the following sentence reports another killing event, which involved different people and took place at a different time and in a different place.

- "*About 23 U.S. soldiers were killed in August 2005 in a suicide attack in Baghdad* "

Based on these observations, in our approach we assume that sentences describing facts of interest to a given schema or set of schemas can be grouped into classes. Each class includes sentences that deliver the same type of information in a similar way (i.e. using similar grammatical constructions). Learning an "extraction rule" for each of these classes would enable the system to extract the desired information from any sentence of interest, from any news feed. A rule can be learned from a sample sentence, after it has been annotated with certain metadata, as we will describe in the following.

In order to deal with different languages two different strategies are possible. The first strategy would consist of translating news feeds in different languages to a single target language (i.e. English) and then apply extraction rules that have been learned for that language. The second strategy would instead consist of learning extraction rules for each language we want to be able to deal with. We adopt the second one because our approach is based on a precise analysis of the grammatical structure of sentences and the current state of the art of Machine Translation does not guarantee sufficiently high quality translations.

Figure 2 shows what the T-REX annotation interface looks like while annotating the sentence "*At least 73 civilians were killed February 1 in simultaneous suicide bombings at a Hilla market*". When describing the annotation process we will consider this sentence as representative of the class including all the sentences listed so far in this section. The annotation process consists of the steps described in the following.

Step 1 The annotator is presented with one or more parse trees for a sample sentence. A parse tree is a tree representation of the grammatical structure of the sentence: non-leaf nodes represent entire phrases (e.g. noun phrases, verb phrases) while leaf nodes represent atomic fragments of the text. Note that the types of phrases may vary with respect to the target language. Figure 3 compares the appearance of the annotation panel with what it would look like while annotating the Spanish sentence "*En el barrio mayoritariamente chi de El Shaab, en el norte de Bagdad, un atentado suicida en un mercado popular dej a el*

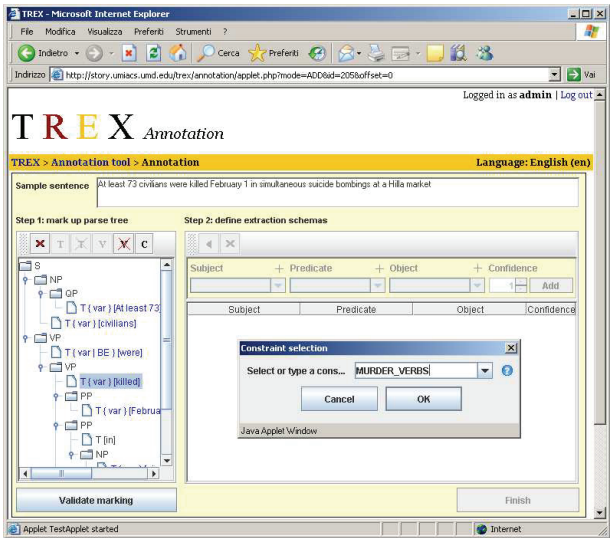
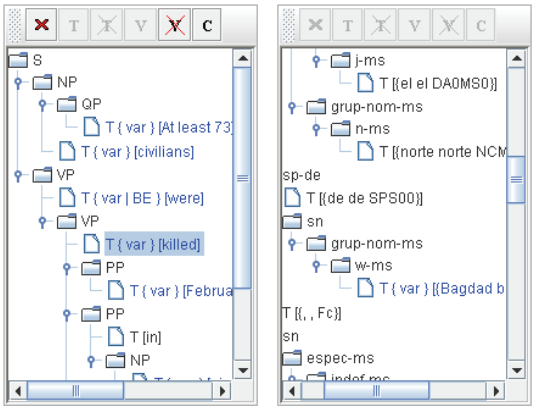


Figure 2: Multilingual Annotation Interface

menos 60 muertos y 25 heridos". The annotation process works in the same exact way except for the different set of phrase types.



a b

Figure 3: Multilingual Annotation Interface

The generation of multiple parse trees is caused by ambiguities in natural language, that may lead to different interpretations, each compatible with the underlying grammar. The annotator selects the interpretation that is most suitable to represent the specific sentence. Figure 4 shows a valid parse tree for the sample sentence.

Step 2 The annotator marks as "variable" all those nodes whose associated text may be different in other sentences of the same class. There are two main reasons for a node to be marked as variable:

1. the node represents a piece of information (e.g. "at least 23" vs. "about 23", "civilians" vs. "U.S. soldiers") that can be extracted from the sentence;
2. the node is a word that helps to discriminate the type of information delivered, but it can be replaced by any synonym (e.g. the verb "killed" lets us know that a killing event took place, but it can replaced by "massacred", "assassinated", "murdered", etc.).

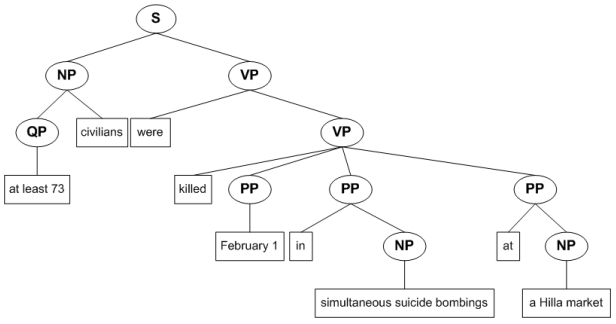


Figure 4: Parse tree of the sample sentence

Please note that when a non-leaf node is marked as variable, this means that the subtree rooted at that node may have an arbitrary structure. Figure 5 shows the parse tree with the attached variable marks.

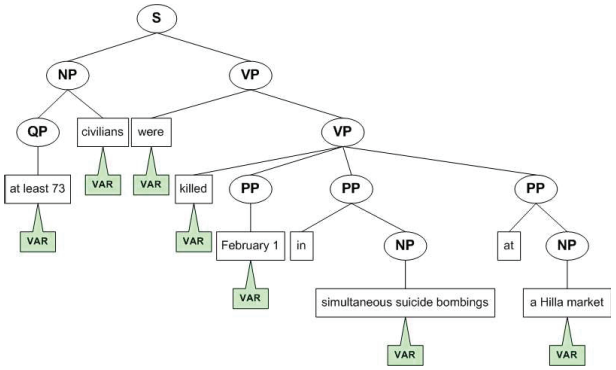


Figure 5: Parse tree of the sample sentence annotated with variable marks

Step 3 If needed, the annotator adds constraints to variable nodes. A variable node can potentially assume any value. However there are cases in which it is desirable to restrict the value of a node within certain boundaries. Possible constraints are:

1. IS_ENTITY, that restricts a noun phrase to be a "named entity" (i.e. the name of a person, organization or place). This restriction is useful for example if we are interested in killing events in which the victim is a precise person rather than a generic group of people;
2. IS_DATE, that restricts a noun phrase to be a temporal expression;
3. X_VERBS, that restricts a verb to be a member of a class X of verbs. In our sample sentence, the constraint $MURDER\_VERBS$ applied to the verb node "killed" would guarantee that any matching sentence describes a killing event, no matter what verb has been used. The classification of verbs we have adopted is

based on the work of Beth Levin (Levin 1993) who classified approximately 3,100 verbs into a hierarchy including more than 300 classes. Verbs are grouped into classes based on meaning and usage. We have expanded the original classification to include more verbs and to further refine the class hierarchy[1];

4. X_NOUNS, that restricts a noun to be a member of a class X of nouns. In our sample sentence we may apply the constraint $ATTACK\_NOUNS$ to the noun node "simultaneous suicide bombings". Similarly we are developing a classification of nouns, that at this times includes about 200 names – mainly denoting relationships among people and/or organizations and events – organized in about 45 classes.

Figure 6 shows the parse tree for the sample sentence annotated with variable marks and constraints.

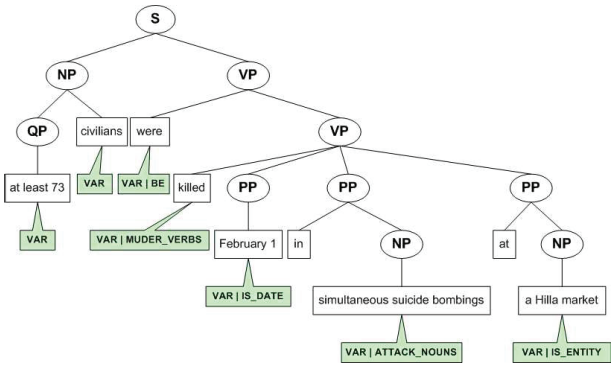


Figure 6: Parse tree of the sample sentence annotated with variable marks and constraints

Step 4 The annotator describes the semantics of the annotated sentence in term of triples, mapping attributes to variable nodes. For instance, the sentence analyzed thus far describes a killing event for which the following four attributes can be instantiated: `victims`, `numberOfVictims`, `location` and `date`. The respective values are "civilians", "at least 73", "Hilla market" and "February 1", each corresponding to a variable node in the parse tree. Figure 7 shows the complete annotation.

Rule Extraction Engine

The Rule Extraction Engine has the purpose of creating an extraction rule from each annotated sentence, taking the RDF schemas into account. The process of extracting a rule once the annotation has been completed is pretty straightforward.

[1]As an example, in the original classification by Beth Levin the class of "verbs of judgement" includes verbs of both positive and negative judgement, just because they are used in the same way. To better meet our needs to represent fine grained information, we have further split this class, including verbs of positive and negative judgement into two different subclasses

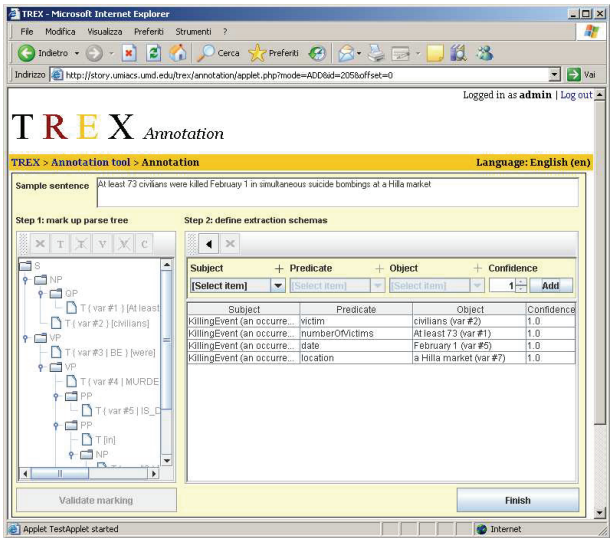


Figure 7: Multilingual Annotation Interface

An extraction rule is of type $Head \leftarrow Body$, where the body represents a condition to be evaluated on a sentence, and the head represents the pieces of information that can be inferred from the sentence if the condition is satisfied.

The process of learning a rule from an annotated sentence consists of the following steps:

abstraction each variable node is assigned a numeric identifier, actual text of variable nodes is removed as well as child nodes of variable nodes; at this step the model becomes independent of the particular sentence;

body definition the body of the rule is built by serializing the parse tree of the annotated sentence in Treebank II Style (Bies *et al.* 1995); the serialized parse trees incorporates variable ids and constraints;

head definition the head is defined as a conjunction of RDF statements, one for each triple defined in the last step of the annotation process.

Figure 8 shows what the extraction rule learned from the sample annotated sentence looks like. Supposing that the system is analyzing a sentence from a news feed, the rule can be read as follows: if the parse tree of the sentence has the same structure of the parse tree in the body of the rule and all the constraints are satisfied (i.e. the auxiliary verb is BE, the main verb belongs to $MURDER\_VERBS$, etc.), then the system can infer that the sentence reports a killing event, where the victim is represented by the value of variable node #2, the number of victims by variable #1, the date by variable #5 and the location variable #7.

Example 1 *Suppose that the system is analyzing the sentence "*About 23 U.S. soldiers were killed in August 2005 in a suicide attack in Baghdad *". The parse tree of this sentence clearly matches the parse tree of the extraction rule in Figure 8, so the following variable assignments can be produced:*

Var#1 = "About 23"; Var#2 = "U.S. soldiers"; Var#3 = "killed"; Var#4 = "August 2005"; Var#5 = "a suicide attack"; Var#6 = "Baghdad".

Var#3 satisfies the constraint BE, Var#4 satisfies the constraint $MURDER\_VERBS$, Var#5 satisfies the constraint IS_DATE, Var#6 satisfies the constraint $ATTACK\_NOUNS$, Var#7 satisfies the constraint IS_ENTITY, thus the following RDF triples can be extracted based on the head of the rule:

```
(KillingEvent9,victim,U.S. soldiers)
(KillingEvent9,numberOfVictims,about 23)
(KillingEvent9,date,August 2005)
(KillingEvent9,location,Baghdad)
```

where `10` is a unique identifier assigned to this particular instance of killing event.

Our corpus currently contains a total of about 550 annotated sentences, leading to 550 extraction rules in all.

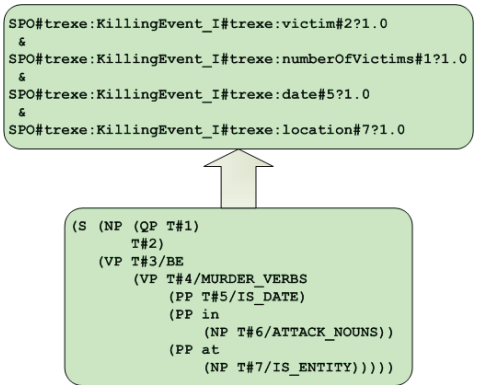


Figure 8: Extraction rule learned from the sample annotated sentence

Rule Matching Engine

The Rule Matching Engine has the objective of extracting RDF triples from global news feeds by matching sentences occurring in news articles against the set of extraction rules.

The RuleMatching algorithm is shown in Figure 9. It takes as input the identifier A of the application for which data is being processed (e.g. "Afghan tribes along the Afghan-Pakistan border", "violent map of the world") and a set $\mathcal{R}$ of rules for extracting the facts of interest to a given schema or set of schemas. The algorithm continuously processes documents as long as there exist documents relevant to application A (lines 1-12). The function $GetDocument(A)$ (line 2) gets a document relevant to A from a repository of cached news articles. This repository is continuously fed by a web crawler that explores a set of assigned news web sites on a daily basis. Each sentence is then matched against the set $\mathcal{R}$ of rules (lines 4-9). If the sentence satisfies the condition in the body of a rule, the statements in the head are instantiated with the values of variable nodes and stored in the database (line 7).

The satisfaction of the condition in the body of the rule is evaluated through the $CompareNodes$ algorithm shown

Algorithm $RuleMatching(A, \mathcal{R})$
Input: An application A and a set of extraction rules.
Output: ARDF triples stored in the database.

```
1  while RelevantDocuments(A) ≠ ∅
2    d ← GetDocument(A)
3    for each sentence s ∈ d
4      for each rule r ∈ R
5        if CompareNodes(s.pTree.root, r.body.root)
6          for each statement m ∈ r.head
7            DBinsert(instantiate(m, s.values))
8          end for
9        end if
10     end for
11   end for
12 end while
```

Figure 9: Algorithm RuleMatching

in Figure 10, that iteratively explore the parse tree of the sentence and the parse tree in the body of the rule and determines if the two match. The algorithm first checks (line 1) if the nodes being compared are of the same type (i.e. the same type of phrase: NP, VP, etc.). If so, it considers the nature of the node. If the node is a variable, then the algorithm checks if its value in the sentence is compatible with the constraints defined on the corresponding node of the model (line 4). If the node is not a variable, but still a leaf node, the algorithm checks if the values of the compared node are identical (line 7). Eventually, if the node is a non-leaf node, the algorithm checks if the nodes being compared have the same number of children and these match pairwise (lines 10-17).

T-REX Implementation

The implementation of our T-REX system consists of several components running on different nodes of a distributed system.

- The Multilingual Annotation Interface is a web-based tool, that is part of the web interface of T-REX. It is implemented as a Java Applet and consists of approximately 2,500 lines of Java code. It has been designed to be used by people with basic computer skills, in order to allow the involvement as annotators of people with diverse backgrounds (e.g. political science).
- The underlying DBMS is PostgreSQL 7.4, installed on a host equipped with a dual-processor dual-core Intel Xeon 5140 CPU running at 2.33 GHz, with 8GB of RAM.
- The Annotated RDF Database System is installed on a virtual host running under VM-Ware ESX on a single core of an identical processor, with 256MB of RAM.
- The Rule Matching Engine that processes news feeds and extract annotated RDF triples is actually a pipeline of several components, some of which are running on a VM-Ware infrastructure.
  - The Crawler is installed on a virtual host running under VM-Ware ESX on a single core of a dual-processor dual-core Intel Xeon 5140 CPU running at 2.33 GHz,

```
   Algorithm CompareNodes(n_sent, n_rule)
   Input: A node n_sent in the parse tree of a sentence and the corresponding node n_rule
   in the body of the rule.
   Output: True if the subtrees rooted at n_sent and n_rule respectively match.
1  if n_rule.type = n_rule, type
2    if isVariable(n_rule)
3      // check if the value of the node is compatible with the constraints
4      if n_sent.value ≅ n_rule.constraints return true else return false
5    else if isConstantLeaf(n_rule)
6      // check the value of a constant leaf node
7      if n_sent.value = n_rule.value return true else return false
8    else
9      // check if child nodes of a non-leaf node match pairwise
10     if N_sent.children.count = n_sent.children.count
11       for each node n ∈ n_rule.children)
12         if ¬CompareNodes(n_sent.getChildAt(n), n) return false
13       end for
14       return true
15     else
16       return false
17     end if
18   end if
19 else
20   return false
21 end if
```

Figure 10: Algorithm CompareNodes

with 1GB of RAM. The implementation consists of approximately 3,200 lines of C# code. The crawler can process an average of 45,000 web pages per day, storing about 10,000 of them (those that are considered relevant for the specific application).

- The English Parser is installed on a virtual host running under VM-Ware ESX on a single core of an identical processor, with 1GB of RAM. The implementation consists of approximately 1,000 lines of C code and relies on the original C implementation of the Link Grammar (Sleator & Temperley 1993). The parser can process about 8,000 documents per day, producing about 300,000 parse trees per day.
- The Spanish Parser is installed on an identical machine. The implementation consists of approximately 1,000 lines of C++ code and relies on the original C++ implementation of the FreeLing language tool suite (Carreras *et al.* 2004). The parser can process about 7,500 documents per day, producing about 240,000 parse trees per day.
- The Extractor is installed on a virtual host running under VM-Ware ESX on a single core of a dual-processor dual-core Intel Xeon 5140 CPU running at 2.33 GHz, with 1GB of RAM. The implementation consists of approximately 3,000 lines of C# code. The extractor can process about 7,000 documents per day, extracting about 18,000 triples per day. Please note that the number of extracted triples will increase significantly as the number of annotated sentences increases.

Related work and conclusions

The aim of Information Extraction (IE) is the extraction and structuring of data from unstructured and semi-structured electronic documents, such as news article from online newspapers (Cowie & Lehnert 1996). Information Extraction involves a variety of issues and tasks ranging from text segmentation to named entity recognition and anaphora resolution, from otology-based representations to data integration.

There is a large body of work in the IE community addressing single issues among those mentioned, and a variety of approaches and techniques have been proposed.

With respect to the issue of named entity recognition, some authors propose knowledge-based approaches (Callan & Mitamura 2002) while others favor the use of statistical models such as Hidden Markov Models (GuoDong & Jian 2003). (Amitay *et al.* 2004) introduces Web-a-Where, a system for locating mentions of places and determining the place each name refers to. In addition, it assigns to each page a geographic focus – a locality that the page discusses as a whole – in order to facilitate a variety of location-based applications and data analysis.

Several attempts have been made to build a comprehensive IE framework. Unfortunately most of such IE tools are domain dependent because they rely either on domain specific knowledge or features such page layouts in order to extract information from text documents.

(Soderland 1997) presents an approach for extracting information from web pages based on a pre-processing stage that takes into account a set of both domain-dependent and

domain-independent layout features. The latter group may include features such as particular formatting styles adopted by given web-sites. Similarly (Gatterbauer *et al.* 2007) focuses on the extraction from web tables, but attempts to be domain independent by taking into account the two-dimensional visual model used by web browsers to display the information on the screen. The obtained topological and style information allows to fill the gap created by missing domain-specific knowledge about content and table templates.

Other efforts are aimed at developing IE capabilities for specific knowledge domains, such as molecular biology (Jensen, Saric, & Bork 2006), and thus use domain specific knowledge to achieve their goals. More general approaches are based on automatic ontology-based annotation mechanisms as the one proposed in (Ding & Embley 2006).

In conclusion, our approach differs significantly from previous approaches because it is a domain-independent framework for information extraction, that does not rely on any feature specific to a given news site or a given knowledge domain. In order to enable domain specific applications, we have developed the capability of targeting the extraction to the instantiation of a schema of interest provided as an input by the user. We have also implemented a complex prototype system that has proved to effectively extract information for different applications and to scale massively as well. Of course, T-REX will benefit and can leverage any domain specific aspects - for example, if it turns out that an extra 50 annotated sentences are available for a given domain, this will certainly increase the accuracy of the T-REX system.

Acknowledgments. Researchers funded in part by grant N6133906C0149, ARO grant DAAD190310202, AFOSR grants FA95500610405 and FA95500510298, and NSF grant 0540216.

References

Amitay, E.; Har'El, N.; Sivan, R.; and Soffer, A. 2004. Web-a-Where: Geotagging Web Content. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 273–280.

Bies, A.; Ferguson, M.; Katz, K.; and MacIntyre, R. 1995. *Bracketing Guidelines for Treebank II Style Penn Treebank Project*. Linguistic Data Consortium.

Callan, J., and Mitamura, T. 2002. Knowledge-Based Extraction of Named Entities. In *Proceedings of the 4th International Conference on Information and Knowledge Management*.

Carreras, X.; Chao, I.; Padr, L.; and Padr, M. 2004. FreeLing: An Open-Source Suite of Language Analyzers. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*.

Cowie, J., and Lehnert, W. 1996. Information extraction. *Communications of the ACM* 39(1):80–91.

Ding, Y., and Embley, D. W. 2006. Using Data-Extraction Ontologies to Foster Automating Semantic Annotation. In *Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06)*, 138.

Gatterbauer, W.; Bohunsky, P.; Herzog, M.; Kroepl, B.; and Pollak, B. 2007. Towards Domain-Independent Information Extraction from Web Tables. In *Proceedings of the 16th International World Wide Web Conference*, 71–80.

GuoDong, Z., and Jian, S. 2003. Integrating various features in hidden markov model using constraint relaxation algorithm for recognition of named entities without gazetteers. In *Proceedings of the International Conference on Natural Language Processing and Knowledge Engineering*, 465–470.

Jensen, L. J.; Saric, J.; and Bork, P. 2006. Literature mining for the biologist: from information retrieval to biological discovery. *Nature Reviews Genetics* 7(2):119–129.

Levin, B. 1993. *English Verb Classes and Alternations: A Preliminary Investigation*. University Of Chicago Press.

Sleator, D. D., and Temperley, D. 1993. Parsing English with a Link Grammar. In *Proceedings of the Third International Workshop on Parsing Technologies (IWPT-93)*.

Soderland, S. 1997. Learning to Extract Text-Based Information from the World Wide Web. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, 251–254.

Subrahmanian, V. S.; Albanese, M.; Martinez, M. V.; Nau, D.; Reforgiato, D.; Simari, G. I.; Sliva, A.; Udrea, O.; and Wilkenfeld, J. 2007. CARA: A Cultural Reasoning Architecture. *IEEE Intelligents Systems* 22(2):12–15.

Udrea, O.; Recupero, D. R.; and Subrahmanian, V. S. 2006. Annotated RDF. In *Proceedings of the 3rd European Semantic Web Conference (ESWC 2006)*, volume 4011 of *Lecture Notes in Computer Science*, 487–501. Springer.

World Wide Web Consortium (W3C). 2004. Resource Description Framework (RDF). http://www.w3.org/RDF/.