published as: S. Linckels, Christoph Meinel: Automatic Interpretation of Natural Language for a Multimedia e-Learning Tool;
Proceedings of the 4th International Conference on Web Engineering (ICWE 2004);
Munich (Germany), 2004, LNCS3140, pp. 435-439.

Automatic Interpretation of Natural Language for a Multimedia E-learning Tool

Serge Linckels and Christoph Meinel

Department for Theoretical Computer Science and New Applications, University of Trier
{linckels, meinel}@TI.uni-trier.de
http://www.informatik.uni-trier.de/~meinel

Abstract. This paper describes the new e-learning tool CHESt that allows students to search in a knowledge base for short (teaching) multimedia clips by using a semantic search engine. We explain the different steps to automatically describe the meaning of the clips with RDF (*Resource Description Framework*). The concept is based on graph theory and retrieval algorithms. Finally, we present rules how a human question can be transformed into a RDF query. Thus, the knowledge base and the query have the same format and can be compared.

1 Our Multimedia E-learning Tool

CHESt (*Computer History Expert System*) is the prototype of a new e-learning tool, see [7] for details. It focuses on three key features: the information is in a multimedia form, the content is split into small *clips* and a semantic search mechanism for information retrieval. We used *Tele-TASK* [1] [2] to record the lessons in order to create one well-structured multimedia stream. The result is a large number of *RealMedia* files that can be played with any compatible software, for example the free *RealOne Player* [5].

Essential in our concept is the length of the stored items in the knowledge base; the duration of the multimedia sequences. The younger the user, the shorter the time during which he/she will concentrate on the information displayed on the screen. Furthermore, it is easier to find the appropriate information inside a small piece of data than for example in an online lesson that lasts 90 minutes. Thus, we divided all our multimedia data into small *clips*. The duration of each clip varies from several seconds to 3 or 4 minutes. Each clip documents one subject or a part of a subject. Together, all the clips of the knowledge base cover one large topic; in our prototype we focus on computer history. We produced more than 300 clips about most important events in computer history. CHESt exists as standalone application (knowledge base and application software on one CD-ROM) and as online application. The later uses a streaming server to transmit the clips to the user's browser.

In this paper we present a retrieval mechanism where the user can enter a complete question. The tool "understands" that question and gives a small list of pertinent clips as answer, or better even just one clip.

N. Koch, P. Fraternali, and M. Wirsing (Eds.): ICWE 2004, LNCS 3140, pp. 435–439, 2004.

2 Describing the Meaning of the Clips

However, before the tool can even try to understand the user's question, it has to "know" what data are stored in the knowledge base. Therefore, we have to add metadata to each clip to describe its meaning. For this purpose we use the *Resource Description Framework* (RDF) [10]. In principle, this is done once, at the moment when the clip is added to the knowledge base. However, the computer can take on a part of this task.

We divided the CHESt knowledge base logically into two classes: clips that describe inventions (things) and clips that describe inventors (persons). Assertion: an invention was invented by one or more inventors. An invention and an inventor can be a *resource* (in our case: a clip) or a *literal* (just a textual information). Every resource is described with properties. An **inventor** has three properties (*predicates*): his name (`vcard:FN`), the year of his birth (`chest:year_birth`) and the year of his death (`chest:year_death`); if still alive, this property is left blank. As you see, we used the W3C recommendation v*Card* namespace property *full name* (FN) [9]. The class **invention** is divided into a number of subclasses to better organize the different resources (for example: Hardware, Software...). We used the *Dublin Core* (dc) namespace [3] to describe an invention with the following properties (*predicates*): its description (`dc:title`), its date of first appearance (`dc:date`) and its inventor (`dc:creator`). The complete CHESt RDF schema can be found at [6].

The next step is to search inside every clip for metadata. We applied an approved approach from the field of computer linguistics: create a dictionary of synonyms for every CHESt RDF element [4] [8]; in one column one will find the RDF elements and in the other column there is a list of natural language synonyms. For example, if we are scanning for `dc:creator`, we are searching for words like *creator*, *builder*, *constructor*, *inventor*, etc. The slides used to create the Tele-TASK clips are converted into pure text files. Then the *stemming process* can begin. All non-words and words with just one letter were eliminated from the generated text files because they have no semantic influence. All words are converted into lowercase and separation characters {, - . ? ! ( ) + * / & @} are replaced by a space. Then, a tree is built from those words, where every node represents one letter (see figure 1). This technique also allows to eliminate all double words. Each node contains the number of words that end with that particular letter.

The dictionary of synonyms is built from that tree. The idea is to regroup words with a similar spelling and thus with the same meaning (for example: build, built, builds). It is impossible to detect automatically all synonyms, because there are words that have a similar spelling, but not the same meaning. The aim of the stemming process is to limit human intervention by proposing clusters of generated synonyms. We got acceptable results with three simple rules. Two words are synonyms only if all three rules match.

- Firstly, the common part (*trunk*) of two words must have a length of at least 4 letters, for example: $trunk(begin, beginning) = \{begin\} \geq 4$.
- Secondly, the remaining and not common part (*tail*) must not be longer than 5 letters, for example: $tail(begin, beginning) = \{ning\} \leq 5$.

- Thirdly, a different letter is only accepted if the common part has at least 3 letters, for example: *trunk*(*begin, began*) = {*beg*} ≥3.

Finally, RDF elements were affected to the concerned clusters, for example the cluster containing the words {*begin, begins, beginning, start, starting*} becomes synonym for `dc:date` and the words {*inventor, builder, constructor, inventors*} are affected to `dc:creator`. The final clustered dictionary is stored for later use (see section 3). The final step consists in scanning through the clips and searching for synonyms for the RDF elements. The result is a RDF/XML serialization for each clip.

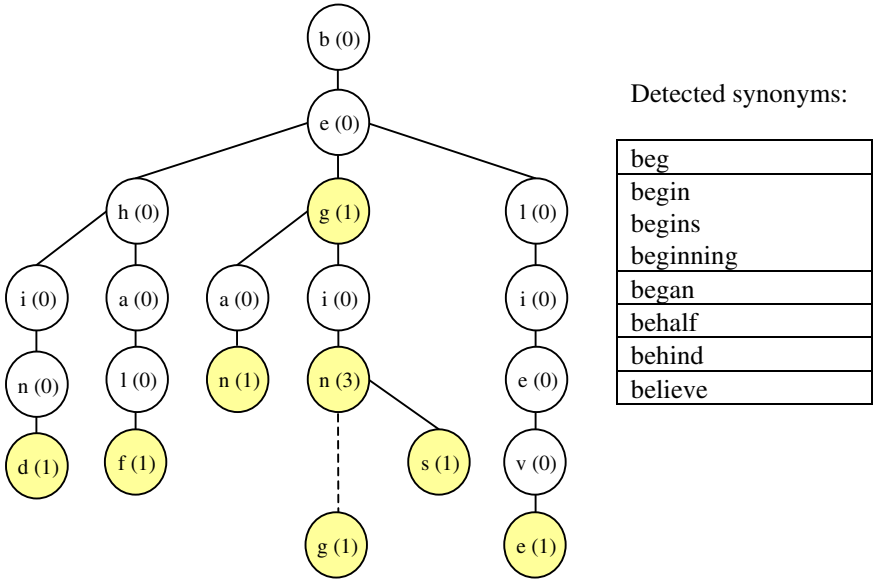

Fig. 1. Example of a generated tree of words. The number in brackets indicates the number of occurrences of the word. If the number is zero, then this node is no final letter. In this generated example, no wrong synonym is found. But one synonym was not clustered: {*began*} should be placed in the cluster {*begin*, *begins*, *beginning*}.

3 Understanding the User

To perform a semantic search, the question entered by the user must be transformed into RDF, in order to have the same structure for the question and for the database. The backbone of our semantic search is an inference engine which transforms a normal sentence (the user's question) into a well-formulated RDF query. For example: "Who invented the very first calculator" should become:

```
SELECT <?x> WHERE <chest:Computer>;<dc:creator>;<?x>
```

We will not describe details about representing RDF data in a database or how to launch a RDF query; see for example [11]. We will focus on the parsing of the sentence and the construction of the RDF query.

Table 1. Illustration of the basic rule for transforming a user's question into a RDF query.

Question	Subject	Predicate	Object
Who built the first calculator?	`chest:Computer` (*calculator*)	`dc:creator` (*built*)	`?x`
What does Zuse invent?	`?x`	`dc:creator` (*invent*)	`chest:Person` (*Zuse*)

Table 2. Illustration of several exceptions for transforming a user's question into a query.

Question	Subject	Predicate	Object
When was Aiken born?	`chest:Person` (*Aiken*)	`chest:year_birth` (*born*)	`?x`
What was the year Aiken died?	`chest:Person` (*Aiken*)	`chest:year_death` (*died*)	`?x`
What does ARPA mean and who founded it?	`chest:Firm` (*ARPA*)	`dc:creator` (*founded*)	`?x`
	`chest:Firm` (*ARPA*)	`dc:title` (*mean*)	`?x`
Who built the ENIAC and the EDVAC?	`chest:Computer` (*ENIAC, EDVAC*)	`dc:creator` (*built*)	`?x`
When did Zuse build his Z3?	`chest:Computer` (*Z3*)	`dc:creator` (*build*)	`?x`
	`?x`	`dc:creator` (*build*)	`chest:Person` (*Zuse*)
What is Linux?	`chest:OS` (*Linux*)		

The transformation of a common formulated sentence into RDF can be summed up by saying that the system has to replace all semantically important words by the RDF corresponding elements and to throw unimportant words away. For the question "Who invented the very first calculator?" the following words were replaced: {*invented*}→ `dc:creator`, {*calculator*}→ `chest:Computer`. All other words will not be considered. The missing part becomes the subject of the query. See table 1 for some general examples. But there are a lot of imaginable exceptions, for example:

- The predicate is not `dc:creator` (see table 2, lines 1+2). In that case, we are not in the basic assertion: "An invention was invented by an inventor", thus the general rule cannot be applied. It is a fact that the missing part must be the object. It is also a fact that the user is not searching for a person or an invention. There are several possible predicates depending on the class-membership of the subject: {`dc:date` and `dc:title`} if the subject is an invention or {`chest:year_birth`, `chest:year_death` and `vcard:FN`} if the subject is an inventor. The parser must choose the right predicate by analyzing the other found synonym(s), for example: words like "born" or "died" indicate a date.
- There is more than one predicate in the sentence. If the predicates are not concurrent then there will only be one query. If there are concurrent predicates (see table 2, line 3) then there will be as many queries as there are different predicates.

- There is more than one subject or object in the sentence. In analogy to the above exception, if the subjects or objects are not concurrent (see table 2, line 4), there will only be one query. If there are concurrent subjects or objects then there will be as many queries as there are different subjects or objects.
- There is no missing part. The question contains a predicate, a subject and an object (see table 2, line 5). This is the most complicated exception to handle. The system must find the best matching clip by associating the different queries.
- There are less than two known parts. In that case, the system lists all resources matching the keywords for the given class (see table 2, line 6).

4 Outlook

The prototype CHESt is tested with a simple keyword search in some selected schools in the summer term of the year 2004. Meanwhile, we are working on the improvement and development of the semantic search engine described in section 3. A prototype is to be tested in a larger pilot project in several schools and universities for the coming winter term (interested schools can contact us). The experience and empiric data that will be collected with the educational tool CHESt should then be the base of further research for a more general semantic search engine. One could imagine developing a generalized interface to access the knowledge base that contains clips of different topics: geography, French vocabulary, irregular English verbs, explanation of HTML tags, biography about famous actors, etc.

References

1. Chen T., Ma M., Meinel Ch., Schillings V.: Tele-TASK, Teleteaching Anywhere Solution Kit. Universität Trier. http://www.tele-task.de/
2. Meinel Ch., Schillings V.: Tele-TASK - Teleteaching Anywhere Solution Kit. In proceedings of ACM SIGUCCS 2002, Providence, USA (2002), pages 130-133
3. Dublin Core Metadata Initiative (DCMI). http://dublincore.org
4. Manning Ch., Schütze H.: Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge London (2003)
5. Real.com: RealOne Player. http://www.real.com/
6. Linckels S.: CHESt namespace. http://www.linckels.lu/chest/elements/1.0/
7. Linckels S., Meinel Ch.: An Application of Semantics for an Educational Tool. In proceedings of IADIS International Conference of Applied Computing 2004, Lisbon, Portugal (2004), pages II-234 - II 239
8. Carstensen K.-U. et al.: Computerlinguistik und Sprachentechnologie. Spektrum Lehrbuch (2001)
9. World Wide Web Consortium: Representing vCard Objects in RDF/XML. http://www.w3.org/TR/2001/NOTE-vcard-rdf-20010222/
10. World Wide Web Consortium: Resource Description Framework (RDF) / W3C Semantic Web. http://www.w3.org/RDF/
11. Karvounarakis G. et al.: RQL, A Declarative Query Language for RDF. In proceedings of ACM WWW2002, Honolulu, USA (2002)