Ontology Construction for Web Services

Aviv Segev[1] and Quan Z. Sheng[2]

[1] Department of Knowledge Service Engineering, KAIST, Daejeon 305-701, Korea
aviv@kaist.edu
[2] School of Computer Science, The University of Adelaide, SA 5005, Australia
qsheng@cs.adelaide.edu.au

Abstract. Ontologies have become the de-facto modeling tool of choice, employed in a variety of applications and prominently in the Semantic Web. Nevertheless, ontology construction remains a daunting task. Ontological bootstrapping, which aims at automatically generating concepts and their relations in a given domain, is a promising technique for ontology construction. Bootstrapping an ontology based on a set of predefined textual sources, such as Web services, must address the problem of multiple concepts that are largely unrelated. This paper exploits the advantage that Web services usually consist of both WSDL and free text descriptors. The WSDL descriptor is evaluated using two methods, namely Term Frequency/Inverse Document Frequency (TF/IDF) and Web context generation. We propose an ontology bootstrapping process that integrates the results of both methods and validates the concepts using the free text descriptors, thereby offering a more accurate definition of ontologies.

1 Introduction

Ontologies are used in an increasing range of applications, notably the Semantic Web, and essentially have become the preferred modeling tool. However, the design and maintenance of ontologies is a formidable process [1]. Ontology bootstrapping, which has recently emerged as an important technology for ontology construction, involves automatic identification of concepts relevant to a domain and relations between the concepts [2]. Previous work on ontology bootstrapping focused on either a limited domain or expanding an existing ontology [3]. In the field of Web services, registries such as the Universal Description, Discovery and Integration (UDDI) have been created to encourage interoperability and adoption of Web services. Unfortunately, UDDI registries have some major flaws [4]. In particular, UDDI registries either are publicly available and contain many obsolete entries or require registration which limits access. In either case, a registry only stores a limited description of the available services. Ontologies created for classifying and utilizing Web services can serve as an alternative solution. However, the increasing number of available Web services makes it difficult to classify Web services using a single domain ontology or a set of existing ontologies created for other purposes. Furthermore, the constant increase in the number of Web services requires continuous manual effort to evolve an ontology.

The Web service ontology bootstrapping process proposed in this paper is based on the advantage that a Web service can be separated into two types of descriptions: i) the Web Service Description Language (WSDL) describing "how" the service should be used and ii) a free text description of the Web service describing "what" the service does. This advantage allows bootstrapping the ontology based on WSDL and verifying the process based on the Web service free text descriptor.

The ontology bootstrapping process is based on analyzing a Web service using three different methods, where each method represents a different perspective of viewing the Web service. In particular, the first method analyzes the Web service from an internal point of view, i.e., what concept in the text best describes the document content. The second method describes the document from an external point of view, i.e., what most common concept represents the answers to the Web search queries based on the WSDL content. Finally, the third method is used to resolve inconsistencies with the current ontology. An ontology evolution is performed when all three analysis methods agree on the identification of a new concept or a relation change between the ontology concepts. The relation between two concepts is defined using the descriptors related to both concepts. Our approach facilitates automatic building of an ontology that could assist in expanding, classifying, and retrieving relevant services, without the prior training required by previously developed approaches.

2 Related Work

The field of automatic annotation of syntactic Web services contains several works relevant to our research. [5] presents a combined approach toward automatic semantic annotation of Web services. The approach relies on several matchers (e.g., string matcher, structural matcher, and synonym finder), which are combined using a simple aggregation function. Machine learning is used in a tool called Assam [6], which uses existing annotation of semantic Web services to improve new annotations. [7] suggests a context-based semantic approach to the problem of matching and ranking Web services for possible service composition. Unfortunately, all these approaches require clear and formal semantic annotations to ontologies.

Ontology evolution has been researched on domain specific Web sites [8]. Noy and Klein [1] defined a set of ontology-change operations and their effects on instance data used during the ontology evolution process. Unlike prior work which was heavily based on existing ontology or domain specific, our work evolves an ontology for Web services "from scratch". A survey on the state of the art Web service repositories [9] suggests that analyzing the Web service textual description in addition to the WSDL description can be more useful than analyzing each descriptor separately. The survey mentions the limitation of existing ontology evolution techniques which yield low recall. Our solution overcomes the low recall using Web context recognition.

3 The Bootstrapping Ontology Model

The bootstrapping ontology model proposed in this paper is based on the continuous analysis of WSDL documents and employs an ontology model based on concepts and relationships [10]. The innovation of the proposed bootstrapping model is the combination of the use of two different extraction methods, TF/IDF and Web based, and the verification of the results using a third method analyzing the external service descriptor. We used these three methods to demonstrate the feasibility of our model. Other more complex methods, from the field of Machine Learning (ML) and Information Retrieval (IR), can also be used to implement the model. However, the straightforward use of the methods emphasizes that many methods can be "plugged in" and that the results are attributed to the model's process of combination and verification.

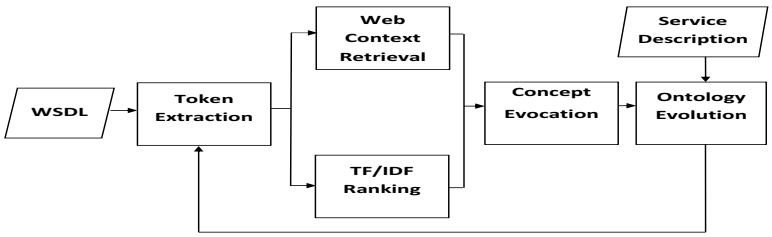

Fig. 1. Web Service Ontology Bootstrapping Process

The overall bootstrapping ontology process is described in Figure 1. There are four main steps in the process. The *token extraction* step extracts tokens representing relevant information from a WSDL document. The second step analyzes in parallel the extracted WSDL tokens using two methods. In particular, *TF/IDF* analyzes the most common terms appearing in each Web service document and appearing less frequently in other documents. *Web context* extraction uses the sets of tokens as a query to a search engine, clusters the results according to descriptors, and classifies which set of descriptors identifies the context of the Web service. The *concept evocation* step identifies the descriptors appearing in both the TF/IDF method and the Web context method. These descriptors identify possible concept names which could be utilized by the ontology evolution. The context descriptors also assist in the convergence process of the relations between concepts. Finally, the *ontology evolution* step expands the ontology as required according to the newly identified concepts and modifies the relations between them. The external Web service textual descriptor serves as a moderator if there is a conflict between the current ontology and a new concept. The relations are defined as an ongoing process according to the most common context descriptors between the concepts. After the ontology evolution, the process continues with the next WSDL. It should be noted that the processing order of WSDL documents is arbitrary.

3.1 Token Extraction

The analysis starts with token extraction, representing each service, $\mathcal{S}$, using a set of tokens called *descriptors*. Each token is a textual term, extracted by simply parsing the underlying documentation of the service. The descriptor represents the WSDL document, formally put as $\mathcal{D}^{\mathcal{S}}_{wsdl} = \{t_1, t_2, \ldots, t_n\}$, where t_i is a token. WSDL tokens require special handling, since meaningful tokens (such as names of parameters and operations) are usually composed of a sequence of words, with the first word lowercase, followed by first letter of other words capitalized (*e.g.*, `getInstitutionNameFromDomain`). Therefore, the descriptors are divided into separate tokens. Figure 2 depicts a WSDL document with the tokens bolded.

The extracted token list serves as a *baseline*. These tokens are extracted from the WSDL document of a Web service that determines whether an email address or domain name belongs to an academic institution. The service is used to illustrate the initial step in building the ontology. All elements classified as *name* are extracted, including tokens that might be less relevant.

3.2 TF/IDF Analysis

TF/IDF is a common mechanism in IR to generate a robust set of representative keywords from a corpus of documents. The method is applied here to the WSDL de-

```
<definitions name="AcademicVerifier"
targetNamespace="http://www.capeclear.com/AcademicVerifier.wsdl" ...>
<message name="isAcademicEmailAddress"><part name="emailAddress">
  <message name="getInstitutionNameFromDomain">
  <message name="getInstitutionNameFromDomainResponse">
  <message name="getInstitutionNameFromEmailAddress">
```

Fig. 2. Initial Processing Example of the Academic Verifier

scriptors. By building an independent corpus for each document, irrelevant terms are more distinct and can be thrown away with a higher confidence. To formally define TF/IDF, we start by defining $freq(t_i, \mathcal{D}_i)$ as the number of occurrences of the token t_i within the document descriptor $\mathcal{D}_i$. We define the term frequency of each token t_i as: $tf(t_i) = \frac{freq(t_i, \mathcal{D}_i)}{|\mathcal{D}_i|}$. We define $\mathcal{D}_{wsdl}$ to be the corpus of WSDL descriptors. The inverse document frequency is calculated as the ratio between the total number of documents and the number of documents which contain the term: $idf(t_i) = \log \frac{|\mathcal{D}|}{|\{\mathcal{D}_i : t_i \in \mathcal{D}_i\}|}$. Here, $\mathcal{D}$ is defined generically, and its actual instantiation is chosen according to the origin of the descriptor. The TF/IDF weight of a token, annotated as $w(t_i)$, is calculated as: $w(t_i) = tf(t_i) \times idf^2(t_i)$.

The token weight is used to induce ranking over the descriptor's tokens. We define the ranking using a precedence relation $\preceq_{tf/idf}$, which is a partial order over $\mathcal{D}$, such that $t_l \preceq_{tf/idf} t_k$ if $w(t_l) < w(t_k)$. The ranking is used to filter the tokens according to a threshold which filters out words with a frequency count higher than the second standard deviation from the average frequency. Figure 3 on the left circle of every concept presents the list of tokens which received a higher weight than the threshold. Several tokens which appeared in the baseline list (see Figure 2) were removed due to the filtering process. For instance, words such as "response" and "get" received below-the-threshold TF/IDF weight, due to their high frequency.

3.3 Context Extraction

We define a context descriptor c_i from domain $\mathcal{DOM}$ as an index term used to identify a record of information, which in our case is a Web service. A weight $w_i \in \Re$ identifies the importance of descriptor c_i in relation to the Web service. For example, we can have a descriptor $c_1 = Academic$ and $w_1 = 42$. A *descriptor set* $\{\langle c_i, w_i \rangle\}_i$ is defined by a set of pairs, descriptors and weights. Each descriptor can define a different point of view of the concept. The descriptor set defines all the different perspectives and their relevant weights, which identify the importance of each perspective.

By collecting all the different view points delineated by the different descriptors we obtain the *context*. A *context* $\mathcal{C} = \left\{\{\langle c_{ij}, w_{ij} \rangle\}_i\right\}_j$ is a set of finite sets of descriptors, where i represents each context descriptor and j represents the index of each set. For example, a context $\mathcal{C}$ may be a set of words (hence $\mathcal{DOM}$ is a set of all possible character combinations) defining a Web service and the weights can represent the relevance of a descriptor to the Web service. In classic IR, $\langle c_{ij}, w_{ij} \rangle$ may represent the fact that the word c_{ij} is repeated w_{ij} times in the Web service descriptor document.

The context recognition algorithm was adapted from [11], which can be formally defined as: Let $\mathcal{D} = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_m\}$ be a set of textual propositions representing a Web service, where for all $\mathcal{P}_i$ there exists a collection of descriptor sets forming the context $\mathcal{C}_i = \{\langle c_{i1}, w_{i1} \rangle, ..., \langle c_{in}, w_{in} \rangle\}$ so that $ist(\mathcal{C}_i, \mathcal{P}_i)$ is satisfied. McCarthy [12] defines a relation $ist(\mathcal{C}, \mathcal{P})$, asserting that a proposition $\mathcal{P}$ is true in a context $\mathcal{C}$. In our case, the adapted algorithm uses the corpus of WSDL descriptors, $\mathcal{D}^{wsdl}$, as propositions $\mathcal{P}_i$ and

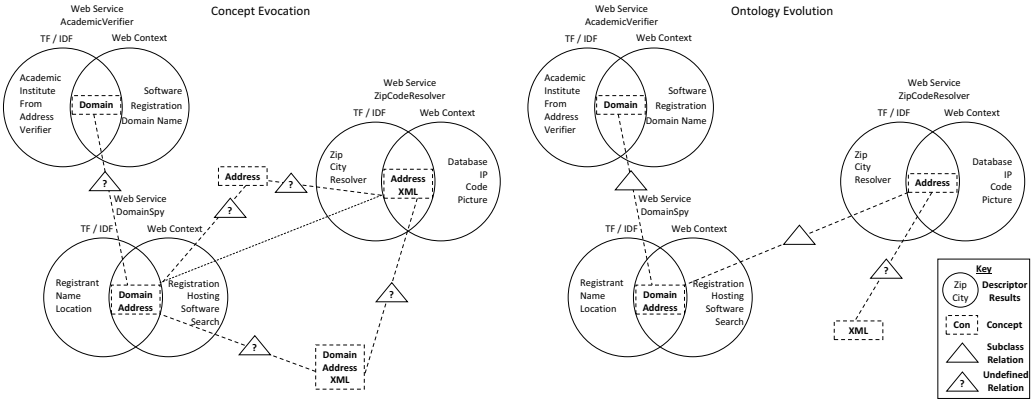

Fig. 3. Example of Web Service Ontology Bootstrapping

the contexts describing the WSDL as descriptors c_{ij} with their associated weight w_{ij}. The context recognition algorithm identifies the outer context $ist(\mathcal{C}, \bigcap_{i=1}^{m} ist(\mathcal{C}_i, \mathcal{P}_i))$.

The context recognition algorithm consists of three main phases: 1) selecting contexts for each text, 2) ranking the contexts, and 3) declaring the current contexts. The result of the token extraction is a list of keywords obtained from the text. The selection of the current context is based on searching the Web for relevant documents according to these keywords and on clustering the results into possible contexts. The output of the ranking stage is the current context or a set of highest ranking contexts. The set of preliminary contexts that has the top number of references, both in number of Web pages and in number of appearances in all the texts, is declared to be the current context and the weight is defined by integrating the value of references and appearances. The input to the algorithm is a stream of information in text format. Figure 3 shows the result of the Web context extraction in the right circle of each concept. The figure shows the context that includes only the highest ranking descriptors which pass the cutoff to be included in the context. For example, *Domain*, *Software*, *Registration*, and *Domain Name* are the context descriptors selected to describe the `AcademicVerifier` service.

3.4 Concept Evocation

Concept evocation identifies a possible concept definition which will be refined in the ontology evolution. The concept evocation is based on context intersection. An ontology concept is defined by the descriptors which appear in the intersection of both the Web context results and the TF/IDF results. We defined one descriptor set from the TF/IDF results, tf/idf_{result}, based on extracted tokens from the WSDL text. The *context*, $\mathcal{C}$, is initially defined as a descriptor set extracted from the Web representing the same document. As a result, the ontology concept is represented by a set of descriptors, c_i, which belong to both sets: $Concept = \{c_1, ..., c_n | c_i \in tf/idf_{result} \cap c_i \in \mathcal{C}\}$.

Figure 3 shows an example of the concept identified by the intersection. For the `AcademicVerifier` Web service, the concept is based on the intersection of both descriptor sets is identified as *Domain*. The concept can consist of more than one descriptor (e.g., `DomainSpy` Web service is identified by the descriptors *Domain* and *Address*). Concepts can be evoked as a result of partial overlapping concepts. This example can be seen by *Address* and the set of *Domain*, *Address*, and *XML*.

A context can consist of multiple descriptor sets and can be viewed as a meta-representation of the Web service. The added value of having such a meta-representation is that each descriptor set can belong to several ontology concepts simultaneously. For example, a descriptor set $\{\langle Registration, 23\rangle\}$ can be shared by multiple ontology concepts (Figure 3) that have interest in domain registration. The different concepts can be related by verifying whether a specific domain exists, domain spying, etc., although the descriptor may have differing relevance to the concept and hence different weights are assigned to it. Such overlap of contexts in ontology concepts affects the task of Web service ontology bootstrapping. The appropriate interpretation of a Web service context that is part of several ontology concepts is that the service is relevant to all such concepts. This leads to the possibility of the same service belonging to multiple concepts based on different perspectives of the service use.

The concept relations can be deduced based on convergence of the context descriptors. The ontology concept is described by a set of contexts, each of which includes descriptors. Each new Web service that relates to the concept adds new context descriptor sets. As a result, the most common context descriptors which relate to more than one concept can change after every iteration. The sets of descriptors of each concept are defined by the union of the descriptors of both the Web context and the TF/IDF results. The *context* is expanded to include the descriptors identified by the Web context, the TF/IDF, and the concept descriptors: $Context_{expanded} = \{c_1, ..., c_n | c_i \in tf/idf_{result} \cup c_i \in \mathcal{C}\}$. For example, in Figure 3, the context of service `AcademicVerifier` includes the descriptors: *Software*, *Registration*, *Domain Name*, *Domain*, *Academic*, *Institute*, *From*, *Address*, and *Verifier*.

The relation between two concepts, Con_i and Con_j, can be defined as the context descriptors common to both concepts, for which weight w_k is greater than a cut off value of a: $Re(Con_i, Con_j) = \{c_k | c_k \in Con_i \cap Con_j, w_k > a\}$. However, since multiple context descriptors can belong to two concepts, the value of a for the relevant descriptors needs to be predetermined. A possible cutoff can be defined by TF/IDF, Web Context, or both. Alternatively, the cutoff can be defined by a minimum number or percent of Web services belonging to both concepts based on shared context descriptors. The relation between the two concepts *Domain* and *Domain Address* in Figure 3 can be based on *Domain* or *Registration*. The example takes a minimum number of appearances in a document as the cutoff of both the TF/IDF and Web Context methods.

3.5 Ontology Evolution

The ontology evolution consists of four steps including: 1) building new concepts, 2) determining the concept relations, 3) identifying relations types, and 4) re-setting the process for the next WSDL document. Building a new concept is based on refining the possible identified concepts. The evocation of a concept in the previous step does not guarantee that it should be integrated with the current ontology. Instead, the new possible concept should be analyzed in relation to the current ontology.

The descriptor is further validated using the textual service descriptor. The analysis is based on the advantage that a Web service can be separated into two descriptions: the WSDL description and a description of the Web service in free text. The WSDL descriptor is analyzed to extract the context descriptors and possible concepts as described previously. The second descriptor, $\mathcal{D}^{\mathcal{S}}_{desc} = \{t_1, t_2, \ldots, t_n\}$, represents the text

1: For each Web service
2: Extract tokens from WSDL
3: TF/IDF_{result} = Apply TF/IDF algorithm to $\mathcal{D}_{wsdl}$
4: $WebContext_{result}$ = Apply Web Context algorithm to $\mathcal{D}_{wsdl}$
5: $PossibleCon_i = TF/IDF_{result} \cap WebContext_{result}$
6: If($PossibleCon_i \subseteq \mathcal{D}_{desc}$)
7: $Con_i = TF/IDF_{result} \cap WebContext_{result}$
8: $PossibleRel_i = TF/IDF_{result} \cup WebContext_{result}$
9: For each concept pair Con_i, Con_j
10: If($Con_i \subseteq Con_j$)
11: Con_i subclass Con_j
12: Else
13: $Re(Con_i, Con_j) = PossibleRel_i \cap PossibleRel_j$

Fig. 4. Ontology Bootstrapping Algorithm

description of the service supplied by the service developer in free text. These descriptions are relatively short and include a sentence or two to describe the Web service. The verification process includes matching the concept descriptors in simple string matching against all the descriptors of the textual service descriptor. We use a simple string-matching function, $match_{str}$, which returns 1 if two strings match and 0 otherwise.

Continuing the example in Figure 3, analysis of the `AcademicVerifier` service yields only one descriptor as a possible concept. The descriptor *Domain* was identified by both the TF/IDF and the Web Context results and matched with a textual descriptor. It is similar for the *Domain* and *Address* appearing in the `DomainSpy` service. However, for the `ZipCodeResolver` service both *Address* and *XML* are possible concepts but only *Address* passes the verification with the textual descriptor. As a result, the concept is split into two separate concepts and the `ZipCodeResolver` service descriptors are associated with both of them.

To evaluate the relation between concepts, we analyze the overlapping context descriptors between different concepts. In this case, we use descriptors which were included in the union of the descriptors extracted by both the TF/IDF and Web context methods. Precedence is given to descriptors which appear in both concept definitions over descriptors which appear in the context descriptors. In our example, the descriptors related to both *Domain* and *Domain Address* are: *Software*, *Registration*, *Domain*, *Name*, and *Address*. However, only the *Domain* descriptor belongs to both concepts and receives the priority to serve as the relation. The result is the relation which can be identified as a subclass, where *Domain Address* is a subclass of *Domain*.

The process of analyzing the relation between concepts is performed after the concepts are identified. The identification of a concept prior to the relation allows in the case of *Domain Address* and *Address* to again apply the subclass relation based on the similar concept descriptor. However, the relation of *Address* and *XML* concepts remains undefined at the current iteration of the process since it would include all the descriptors that relate to `ZipCodeResolver` service. The relation described in the example is based on descriptors which are the intersection of the concepts. Basing the relations on a minimum number of Web services belonging to both concepts will result in a less rigid classification of relations. The process is performed iteratively for each additional service which is related to the ontology. The iterations stop once all the services are analyzed. Alternatively, an ontology administrator can decide to suspend the ontology evolution at any given time.

To summarize, we give the ontology bootstrapping algorithm in Figure 4. The first step is extracting the tokens from the WSDL for each Web service (line 2). The next

step is applying the TF/IDF and Web Context to extract the result of each algorithm (lines 3-4). The possible concept, $PossibleCon_i$, is based on the intersection of tokens of the results of both algorithms (line 5). If $PossibleCon_i$ tokens appear in the document descriptor, $\mathcal{D}_{desc}$, $PossibleCon_i$ is defined as concept, Con_i. The union of all token results is $PossibleRel_i$ for concept relation evaluation (lines 6-8). Each pair of concepts, Con_i and Con_j, is analyzed for whether the token descriptors are contained in one another. If yes, a subclass relation is defined. Otherwise the concept relation can be defined by the intersection of the possible relation descriptors, $PossibleRel_i$ and $PossibleRel_j$ (lines 9-13).

4 Conclusion

This paper proposes an approach for bootstrapping an ontology based on Web service descriptions. The approach analyzes Web services from multiple perspectives and integrates the results. Web services usually consist of both WSDL and free text descriptors. This allows bootstrapping the ontology based on WSDL and verifying the process based on the Web service free text descriptor. The approach enables the automatic construction of an ontology without the prior training required by previously developed methods. As a result, ontology construction and maintenance efforts can be substantially reduced. Our ongoing work includes further performance study of the proposed ontology bootstrapping approach. We plan to apply the approach in other domains in order to examine the automatic verification of the results.

References

1. Noy, N.F., Klein, M.: Ontology Evolution: Not the Same as Schema Evolution. Knowledge and Information Systems **6**(4) (2004) 428–440
2. Ehrig, M., Staab, S., Sure, Y.: Bootstrapping Ontology Alignment Methods with APFEL. In: Proc. of 4th Intl. Semantic Web Conference (ISWC'05), Galway, Ireland (2005)
3. Zhang, G., Troy, A., Bourgoin, K.: Bootstrapping Ontology Learning for Information Retrieval Using Formal Concept Analysis and Information Anchors. In: Proc. of 14th Intl. Conference on Conceptual Structures (ICCS'06), Aalborg University, Denmark (2006)
4. Platzer, C., Dustdar, S.: A Vector Space Search Engine for Web Services. In: Proc. of the 3rd European Conference on Web Services (ECOWS'05), Växjö, Sweden (2005)
5. Patil, A., Oundhakar, S., Sheth, A., Verma, K.: METEOR-S Web Service Annotation Framework. In: Proc. of the 13th Intl. Conference on World Wide Web. (2004)
6. Heß, A., Johnston, E., Kushmerick, N.: ASSAM: A Tool for Semi-automatically Annotating Semantic Web Services. In: Proc. of Intl. Semantic Web Conference (ISWC'04). (2004)
7. Segev, A., Toch, E.: Context-Based Matching and Ranking of Web Services for Composition. IEEE Transactions on Services Computing **2**(3) (2009) 210–222
8. Davulcu, H., Vadrevu, S., Nagarajan, S., Ramakrishnan, I.: OntoMiner: Bootstrapping and Populating Ontologies From Domain Specific Web Sites. IEEE Intelligent Systems **18**(5) (2003) 24–33
9. Sabou, M., Pan, J.: Towards Semantically Enhanced Web Service Repositories. Web Semantics **5**(2) (2007) 142–150
10. Gruber, T.R.: A Translation Approach to Portable Ontologies. Knowledge Acquisition **5**(2) (1993) 199–220
11. Segev, A., Leshno, M., Zviran, M.: Context Recognition Using Internet as a Knowledge Base. Journal of Intelligent Information Systems **29**(3) (2007) 305–327
12. McCarthy, J.: Notes on Formalizing Context. In: Proc. of the 13th Intl. Joint Conference on Artificial Intelligence (IJCAI'93), Chambéry, France (1993)