Lifting XML Schemas to Ontologies - The concept finder algorithm

Philipp Kunfermann, Christian Drumm

SAP Research Center CEC Karlsruhe
SAP AG
philipp.kunfermann@gmail.com
christian.drumm@sap.com

Abstract. In this paper we will present the Concept Finder algorithm. This algorithm is capable of creating mappings between the elements of a XML Schema and the concepts of an existing ontology. Furthermore we present results of a preliminary evaluation where real world schemas from the area of B2B communication were mapped to different ontologies using this algorithm.

1 Introduction

The data mediation problem in the context of web services is concerned with the transformation of a source message M_S which adheres to a source message schema S_S into a target message M_T that adheres to a target message schema S_T. To solve a given mediation problem a mapping needs to be created based on the source and the target message schema. In general the creation of such a mapping in very complex, making the task of developing such transformations very strenuous and error prone [1].

Semantic Web Service (SWS) are seen as the next evolutionary step after Web Service. SWS use ontologies to annotate the used data formats. Mappings between different message formats are in this context created on the semantic rather than on the syntactic level. In order to enable data mediation on the semantic level for existing web service and to bootstrap the semantic annotation of web services we will present an algorithm to relate XML schemas [2] to an existing ontology.

The remaining of the paper is organized as follows. First we briefly introduce the lifting problem. After that we present the algorithm we developed and explain its functionality using a simple example. Finally we present the result achieved by our approach when applied to real world schemas and ontologies.

2 Lifting

As mentioned in the introduction there are different problem areas in the context of SWSs that require the linking between purely syntactical data representation and ontologies. In the remaining of this paper we will call the process of relating syntactical data representations to ontologies *Lifting*.

Definition 1 (Lifting) *Lifting is the process of semantically annotating a source schema S_S with an ontology O.*

Note that this definition doesn't restrict the nature of the input schemas used for the lifting process. Furthermore based on the previous definition 3 different approaches to the lifting problem are possible (see Fig. 1):

– The most basic approach is to create a new ontology based on the source schema and use that ontology for the annotation
– If an existing ontology should be used for the annotation, a mapping between the source schema and the target ontology needs to be created
– Finally a combination of the first two approaches could be used by creating a new ontology from the source schema and by than mapping this ontology to an already existing one. For this approach existing ontology mapping techniques could be reused.

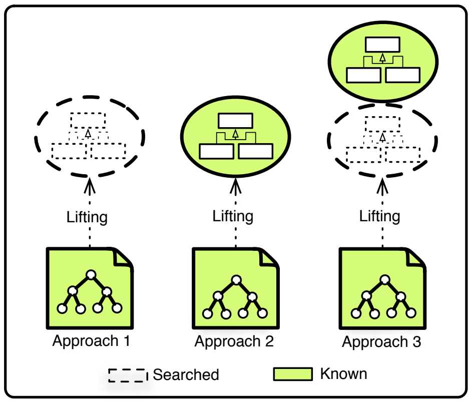


Fig. 1. Different possible approaches to the lifting problem.

In this paper we will present a solution for the second approach describe above. Our solution, the *Concept Finder* algorithm, is capable of identifying mappings between elements of S_S and concepts in O_T. We have chosen that approach for two reasons. Firstly we want to annotate existing WS in order to enable their usage in a SWS environment. For this purpose we need the possibility to easily relate the XML Schemas describing their in and output messages to the domain ontologies. Secondly our approach is capable of integrating different schemas by relating them to a given ontology. This simplifies the integration of existing services using the ontology as intermediate connector.

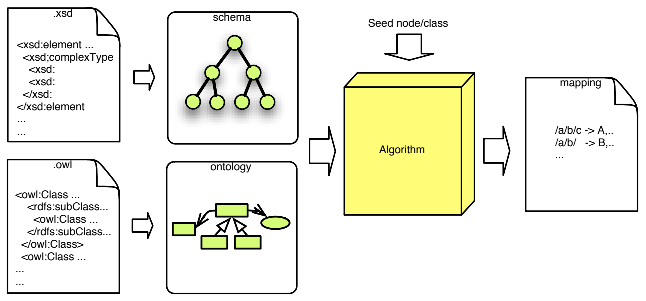
3 The Concept Finder Algorithm

For the discussion of the Concept Finder algorithm presented in this paper we have chosen to restrict the supported input data formats to XML for the source schemas S_S and to OWL [3] for the target ontology O_T. However the Concept Finder algorithm can easily be extended to use other input formats for both, S_S and O_T.

3.1 Overview

Figure 2 shows a high level overview the Concept Finder algorithm. Creating a mapping between a source schema and a target ontology consists of three steps:

1. In the first step the user needs to select the source schema S_S and the target ontology O_T. The Concept Finder algorithm parses the two files and creates an internal representation of them.
2. In the next step the user is presented with a graphical representation of both, S_S and O_T. The user now needs to select a *seed node* in S_S and a matching *seed concept* in O_T. This information is necessary in order to give the Concept Finder algorithm a starting point for exploring the ontology.
3. In the last step the algorithm computes a list of mappings between S_S and O_T. Details on how this is done will be given in the subsequent sections.



```
<xsd:element ...
  <xsd;complexType
    <xsd:
    <xsd:
  </xsd:
</xsd:element
...
...
```

```
<owl:Class ...
    <rdfs:subClass...
      <owl:Class ...
    </rdfs:subClass...
  </owl:Class>
  <owl:Class ...
...
...
```

```
/a/b/c -> A,..
/a/b/   -> B,..
...
```

Fig. 2. High level overview of the Concept Finder algorithm

3.2 Details

Starting with the seed node and the seed concept, the algorithm compares the elements of the schema with relationships in the ontology. In order to do so, it navigates through the schema and the ontology in different ways. The structure of the schema is traversed depth first while the ontology is traversed based on the matches that were found with the compared schema elements.

The seed node and seed concept are a schema respectively ontology element that have to be semantically corresponding. They represent the first match of our algorithm and define the context in which the algorithm operates in the schema as well as in the ontology. In order to find the next match, the Concept Finder algorithm compares every child of the seed node[1] with the relationships of the last matching concept (in the first step this is the seed concept). Firstly, all subclasses of the seed class (find subclass match) are compared. If no match is found, the algorithm compares all properties of the seed class (find property match). If still no match is found for the current element, the child nodes of the current element are explored in order test if a match can be derived (derived match).

A *similarity test* is used to determine if an elements of the S_S and an element of the O_T are matching based on their names. It is processed in three steps. In the first step a normalized levenshtein distance is calculated between the two names and if it is above a given threshold parameter, the elements are accepted as matching. If this is not the case, the name of the schema element is in a second step levenshtein compared with synonyms of the ontology element. In order to find these synonyms *WordNet*[4] is used as an external knowledge source. If still no match is found, the names of the compared elements are tokenized in a third step. These tokens are compared as in the first step and a new metric is calculated based on the number of existing and matching tokens. Again a threshold is applied in order to determine if two elements are matching. If the elements do not match the algorithm passes to the next schema element that will be compared with the relationships in the ontology.

After investigating possible mismatches between schema and ontology models, we discovered that the most important mismatches originate from the developer's freedom on how and what he is modeling using a concrete formalism. Because of the importance of these human caused mismatches, similarity measures have to be concentrated on features that are intuitive for human beings. While humans do hardly agree on the data-type, restrictions or even how many relationships are relevant for a certain concept, names contain important semantics, understandable to every person speaking the same language. Even though the developer may use different terminology, he will use a name that specifies the semantic of what he wants to represent. By using synonyms, we test the elements against a wide range of semantically identical concepts and therefore find correct match, if existing, with high probability. Nevertheless, from natural language processing (NLP) it is known that single words may cause ambiguities. This is prevented by looking at the context of the word. While NLP tries to interpret the sentence in which a word is presented, the context of the names in our case is found in the structure of the used models (e.g. branch of the

[1] In the following we will call the element that is compared by the algorithm current element.

schema tree). Therefore, Concept Finder navigates through the two models and compares only elements that are in the same context.

For every current schema element the algorithm does first try to *find a subclass match*. Therefore all subclasses - not only the direct ones - of the last matched concept are used. This last matched concept semantically corresponds to the parent element of the current schema element. If the current element is matching one of the compared subclasses, the relationship with the parent element is identified as an inheritance relationship. If no such match is found, the relationships of the last matching concept will be compared.

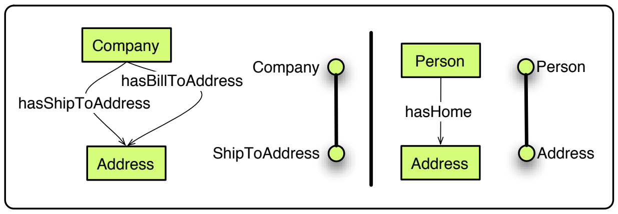


Fig. 3. The object property conflict.

Trying to *find property matches*, the Concept Finder algorithm compares all relationships, data-type properties and object properties, of the last matching concept with the current element. This comparison is more difficult for object properties than for datatype properties, as with object properties both, the relationship itself as well as the range concept, have a semantics carrying name and therefore both of them have to be taken into consideration for the comparison. Figure 3 illustrates that if only the range would be tested, the similarity for $(p_{hasShipToAddress}, c_{Address})$[2] and $(p_{hasBillToAddress}, c_{Address})$ would be the same with respect to the used similarity test while only the first match is semantically correct. If in the opposite case only the property would be tested, $(p_{hasHome}, c_{Address})$ would not be accepted as a match even though it correspond perfectly. In order to prevent that, the similarity test compares both and calculates the similarity as a mixed and weighted metric.

If no property match was detected, the algorithm tries to capitalize on the knowledge of inheritance contained in the ontology and *derive a match* by continuing to explore the children of the current element. They are still compared with the relationships of the last matching concept, knowing that subclasses inherit all relationships of its parents. If a certain number children of the current

[2] When describing the algorithm using examples we will use the following notation: c_{name} and p_{name} denote a concept or a property in the ontology with a given name; e_{name} denotes a element in the schema with a given name.

element match with relationships of the last matching concept, the probability for the current element to be corresponding to a subclass is high.

The algorithm navigates in this way through the whole schema tree while it visits only the relevant concepts of the ontology matching with the schema elements. This context based navigation makes sense because of the ontologies nature. As ontologies are used to define concepts and not only to structure information, we can suppose that if a relationship between two concepts exists in the real world, they are also represented in the ontology and therefore its context is explicitly defined. As in the ontology development no general rules exist on how elaborated a ontology has to be, the algorithm requires to have a meaningful and sufficiently elaborated ontology as input.

4 Example

In order to illustrate how the algorithm works in more detail, we will apply it to the schema and ontology excerpt shown in figure 4 and describe the different execution steps that match these two.

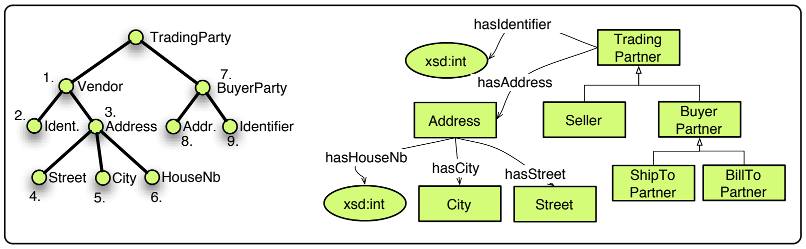


Fig. 4. Applying the algorithm.

- Starting with the first match chosen by the user (seed node and seed class) $e_{TradingParty} \rightarrow c_{TradingPartner}$, $e_{\boldsymbol{TradingParty}}$'s children, e_{Vendor} and $e_{BuyerParty}$ are explored in order to match them to the ontology[3].
- In the first step a match the algorithm tries to find a match for $e_{\boldsymbol{Vendor}}$. Comparing the subclasses and properties of $c_{TradingPartner}$ with e_{Vendor} would not result in any match[4].

[3] Figure 5 demonstrates the algorithm's execution. The current seed elements are shown in blue/dark-grey, the ones that were already successfully compared are shown squared and the unsuccessfully compared ones are show yellow/light-grey.

[4] This is the case if WordNet is not used or does not contain "Vendor" as a synonym for "Seller". We will assume this match is not found for illustrating the further execution steps

- Therefore,Concept Finder starts to explore the e_{Vendor}'s child $e_{\boldsymbol{Identifier}}$. No subclass match is found but a correspondence with the trading partner's identifier $e_{Identifier} \rightarrow p_{hasIdentifier}$.
- Because of the match $e_{\boldsymbol{Address}} \rightarrow (p_{hasAddress}, c_{Address})$, the exploration continues to compare the children of $e_{Address}$ with the range class of the match $(c_{Address})$.
- $e_{\boldsymbol{Street}} \rightarrow (p_{hasStreet}, c_{Street})$, $e_{\boldsymbol{City}} \rightarrow (p_{hasCity}, c_{City})$ and $e_{\boldsymbol{HouseNb}} \rightarrow p_{hasHouseNb}$ are found but cannot be further explored since they are leaf nodes.
- The algorithm selects e_{Vendor} and based on the matches found for its child node he derives that e_{Vendor} could correspond to a subclass of $c_{TradingPartner}$. A match $e_{Vendor} \rightarrow c_{TradingPartner}$ is stored with the special remark that this is a derived subclass match[5].
- Now the algorithm selects $e_{\boldsymbol{BuyerParty}}$,the next child node of $e_{TradingPartner}$ where the match $e_{BuyerParty} \rightarrow c_{BuyerPartner}$ is stored.

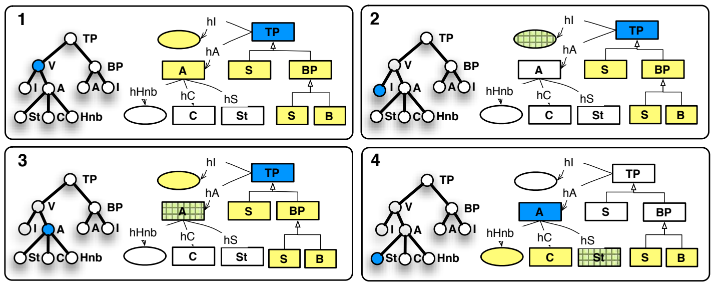


Fig. 5. The beginning proceeding of the algorithm.

In continuation "Address" and "Identifier" are matched after the same procedure until all elements of the schema tree have been explored.

5 Evaluation

We evaluated the Concept Finder algorithm using four scenarios based on different real world schemas and different ontologies. However, as these evaluations using large real world schemas and large ontologies are very time consuming and

[5] Note that it would easily be possible to test for all subclasses contained in the ontology, if children of e_{Vendor} match to properties of them. Hereby the exact subclass match could be found automatically. As this demands much more computational effort it was not implemented in the prototype and therefore the user has to assign the exact subclasses manually at the moment.

strenuous, we were not able to perform a comprehensive evaluation. Instead we created three evaluation scenarios based on the following real world schemas:

- `sap-order.xsd`
- `catalog.xsd`

The `sap-order` schema is a schema developed by SAP describing a purchase order in the well known order to invoice process [5]. The second schema, the `catalog.xsd` describes a product catalog and is part of the BMECat [6] standard. In addition to the schemas two different ontologies where used in the evaluation scenarios. The first one, which will in the following be called *Lifting Ontology (LO)* was manually developed after studying the SAP order to invoice process. The second one, the *Business Data Ontology (BDO)* [7] is an ontology developed as part of the DIP project [8]. It is based on the UBL [9] standard and describes the domain of business-to-business communication.

Based on these schemas and ontologies the following scenarios where developed for the evaluation:

- **Scenario 1:** *sap-order.xsd* $\rightarrow$ *LO*. Our first scenario uses similar inputs as the LO is based on the set of schemas which `sap-order.xsd` is part of.
- **Scenario 2:** *sap-order.xsd* $\rightarrow$ *BDO*. The BDO covers a bigger domain and is much more complete in sense of number of concepts than the LO ontology. In this scenario the only dependency between the schema and the ontology is the domain of interest. They have been created independently and for different purposes.
- **Scenario 3:** *catalog.xsd* $\rightarrow$ *LO*. For the third scenario the schema originates from a different domain than the used ontology, namely the exchange of product catalogues.
- **Scenario 4:** *catalog.xsd* $\rightarrow$ *BDO*. This scenario is similar to the third scenario. Only the used target ontology differs.

Using the metrics used in [1] we achieved the results presented in table 1. The table shows for each of the four scenarios described above two results. This is due to the fact that we evaluated the Concept Finder algorithm using two operation modes. The first row of values show the results achieved when running the algorithm in fully automatic mode whereas the second row shows the results when running the algorithm semi-automatically.

5.1 Discussion

The results of the scenarios 1,3 and 4 show, that the Concept Finder algorithm generally achieves very good results. Even for schemas that only partly overlap with the domain of the ontology, algorithm achieves and overall result between 0.75 and 0.8 in automatic mode and between 0.85 and 0.92 in semi-automatic mode.

In the second scenario, the algorithm achieves only poor results running automatically. The results in this scenario significantly improve if the algorithm

Table 1. The results of the evaluation of the Concept Finder algorithm

	Precision	**Recall**	**Overall**
Scenario 1*a*	0.746	0.842	0.898
Scenario 1*b*	0.831	0.858	0.970
Scenario 2*a*	-0.248	0.050	0.143
Scenario 2*b*	0.375	0.594	0.731
Scenario 3*a*	0.643	0.857	0.800
Scenario 3*b*	0.786	0.857	0.923
Scenario 4*a*	0.400	0.600	0.750
Scenario 4*b*	0.500	0.600	0.857

is run semi-automatically. The reason for the poor results in automatic mode is, that if the ontology is not complete enough, it is possible that by not finding a matching concept for the current element, the algorithm is mislead. Thus, if the comparison point (current class) in the ontology does not anymore correspond to what the element in the schema represents, the algorithm may not recover and will only produce false matches, leading to poor results.

Therefore the overall results of our experiments show are twofold. On the one hand, even in the full automatic mode, the algorithm performs very well for most inputs. On the other hand it is possible that the Concept Finder algorithm is mislead and may perform bad if the structures of the source schema and the target ontology are too different.

An important observation of the evaluation is that a semi-automatic lifting in general seems to perform very good in terms of precision, recall and manual effort and seems to be a very promising approach.

6 Summary and Outlook

In this paper we have presented a new algorithm for lifting existing XML Schema to ontologies. The first evaluation of this algorithm using real world schemas originating from the area of B2B communication show very promising results.

The current implementation of the algorithm is based on a very simple linguistic algorithm to identify possible correspondences between the source schema and the target ontology. If a set of matchers, similar to the ideas presented in [1] would be used, accuracy of the created mappings could be further improved. Furthermore $1-n$ and $m-1$ correspondences between schema elements and ontology concepts need to be taken into account. Finally a improvement of the user interface for the semi-automatic creation of liftings could improve the quality of the results and minimize the necessary user effort.

References

1. Do, H.H., Rahm, E.: COMA - a system for flexible combination of schema matching approaches. In: Proc. 28th VLDB Conference. (2002)

2. W3C: XML schema. Online (2001) http://www.w3.org/XML/Schema.
3. McGuinness, D., van Harmelen, F.: Owl web ontology language overview. http://www.w3.org/TR/owl-features/#s3.4 (2004) [Online; accessed 6-Sept-2005].
4. Princeton University: Wordnet - a lexical database for the english language. http://wordnet.princeton.edu/index.shtml (2005) [Online; accessed 3-Jul-2005].
5. SAP AG: Order schema. http://sap.com/xi/EBP (2002)
6. eBusiness Standardization Commitee: Bmecat. http://www.bmecat.org (2005) [Online; accessed 25-Aug-2005].
7. Nagypal, G., Lemcke, J.: D3.3 a business data ontology, wp3: Service ontologies and service description. - **D3.3** (2005) 141
8. DIP: Data, information, and process integration with semantic web services. online (2004) http://dip.semanticweb.org/.
9. OASIS: UBL. Online (2003) http://www.oasis-open.org/committees/ubl/.