Cost-Effective Semantic Annotation of XML Schemas and Web Service Interfaces

Peep Küngas, Marlon Dumas
Institute of Computer Science
University of Tartu
Tartu, Estonia
{peep.kungas,marlon.dumas}@ut.ee

***Abstract*—Research in the field of semantic Web services aims at automating the discovery, selection, composition and management of Web services based on semantic descriptions. However, the applicability of many solutions developed in this field is hampered by the costs associated with semantically annotating large repositories of Web services. To overcome this gap we propose a practical method for semantically annotating collections of XML Schemas and Web service interfaces. We have evaluated this method on a large repository of governmental Web services. The evaluation shows that relatively simple techniques are surprisingly cost-effective, saving hundreds of man-hours of semantic annotation effort. Moreover, the proposed method does not assume the availability of a pre-existing ontology or controlled vocabulary. Instead, the space of annotations is dynamically built during the annotation process.**

Keywords - semantic annotation; Web Services interfaces; cost-effective annotation methods; XML Schemas; WSDL; SA-WSDL

I. INTRODUCTION

The potential uses of semantically-annotated content and service descriptions, for example in the area of Web service discovery and composition, are widely acknowledged [GVS+06, NVS+06, RKM06, BH08]. However, collecting such semantic annotations is a major bottleneck. Manual approaches to semantic annotation give high precision, but they are generally too costly [EMS+00]. In some scenarios, this cost can be spread across large communities (e.g. through collaborative tagging), but this approach is only applicable when there is an incentive for large communities to come together. On the other hand, several automated semantic annotation methods have been proposed. These automated methods are often based on shallow natural language processing techniques (e.g. morphological analysis, named entity extraction, part-of-speech tagging), taxonomy-based disambiguation and/or machine learning. However, for these approaches to achieve high precision, they require domain-dependent fine-tuning. Some approaches additionally require a pre-existing taxonomy or domain ontology, as well as the availability of linguistic knowledge bases, which are sometimes not available.

In this paper we propose a practical methodology for semi-automated annotation of web service repositories. The aim of the methodology is to enable developers to cost-effectively annotate corpora of WSDL interfaces in order to generate model references (i.e. references to vocabularies, taxonomies or ontologies). These model references are encoded in SA-WSDL (Semantic Annotations for WSDL and XML Schema) [FL07]. Our proposal abstracts away from the specific type of model references used as semantic annotations. These references can point to ontologies encoded in OWL [OWL04], to UML models created according to OUP (Ontology UML Profile [ODM]) or any other model addressable via Unified Resource Identifiers (URIs). Moreover, the proposed approach does not assume that the model (vocabulary, ontology or taxonomy) is available upfront. Instead, the model is incrementally constructed by the annotator(s) as part of the annotation process.

While simple, the proposed methodology has proved to be cost-effective. We applied the methodology to semantically annotate the Estonian governmental Web services repository [K+05], containing circa 60 information systems and around 1000 Web service operations. In this paper, we analyze the cost of creating semantic annotations for this repository using the proposed methodology, and compare it with the estimated effort taken by a fully manual annotation approach. The evaluation suggests that simple methods are great effort savers, i.e. the 80/20 principle applies. Furthermore, the approach can be combined with existing automated annotation solutions either for fine-tuning or bootstrapping large-scale annotation projects.

The rest of the paper is structured as follows. In Section II we describe our semantic annotation methodology. In Section III we present an empirical evaluation of the methodology. In Section IV we review related work while in Section V we draw conclusions and discuss future work.

II. METHODOLOGY

The specific aim of the methodology is to attach semantic annotations (encoded as SA-WSDL model references) to the leaf node elements of an XML schema. In other words, model references are attached only to XML schema elements that have either a built-in XSD type or a "*simpleType*." The rationale for this choice is that leaf elements represent maximally fine-grained data objects (e.g. an apartment number in a customer address). Once these fine-grained elements have been annotated, the resulting annotations can be propagated to coarser-grained elements as required (i.e. to complex types, WSDL message parts, operations and interfaces). This choice does not imply that we only analyze leaf nodes. In fact, the proposed methodology exploits the hierarchical structure of WSDL

interfaces and XML schemas in order to extract contextual information that is used to derive semantic annotations for leaf nodes.

The proposed methodology is inspired by UPON [NMN09], an incremental methodology for ontology building. Accordingly, the methodology includes *cycles*, *phases*, *iterations*, and *tasks*. Cycles are regulated by the evolution of Web services descriptions – every time a WSDL or related XSD documents changes, a new cycle is started, which ends with a complete annotation of the new or modified Web service descriptions. Each cycle consists of 3 phases, namely *annotation of most outstanding elements*, *annotations of most recurrent elements*, and *annotation of other elements*. Each phase is further subdivided into an unlimited number of iterations. Each iteration includes three tasks: analysis of element names, design of heuristic rules and application of heuristic rules. This process is sketched in Figure 1.

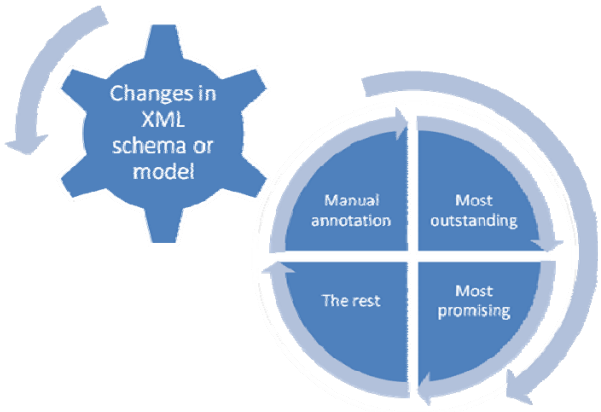

Figure 1. Annotation process.

In each cycle, the annotator proceeds as follows:

- Determine the WSDL and XSD documents to be annotated;
- Build a list of elements to be annotated. First, extract the leaf element nodes of the XML schema. Next, construct a list with each element name, its location (represented by an XPath expression), the URL of the parsed document and any associated documentation. Note that instead of XPath, we could use XSD component designators, which provide an URI-compliant way of referring to elements in an XML schema.
- Optionally, import a data model, ontology or other model, containing the entity definitions that will be used to annotate the XML schema elements;
- Create annotation heuristics;
- Apply the heuristics and export the constructed mappings as SA-WSDL references within the original WSDL/XSD documents;
- Manually create annotations for elements that cannot be annotated by means of the heuristics;
- Extend model, if needed, to cover new entities.

The core phases of each cycle are those related to the manual creation of annotation heuristics. These are summarized below:

- *Create annotation heuristics for the top-10% most "outstanding" element names*. Identify the top 10% most frequently occurring element names and present these to the annotator. The annotator then defines annotation heuristics for each of these "outstanding" elements. Typically, these annotation heuristics cover a significant number of elements in the schema. In this phase, the annotation heuristics are designed to match exact element names (e.g. "customer_name", "customer_address", "id_code", etc.). In the next phase, we will seek to define heuristics that match syntactically similar element names.
- *Create annotation heuristics for the top-50% most "promising" elements*. After removing the most outstanding elements from the "to-do" list, identify the top 40% most recurrent element names in terms of number of occurrences. The frequency of element names processed in this phase is much lower than in the previous phase. Accordingly, the annotation heuristics defined in this phase need to be more generic: instead of defining heuristics that match exact element names, we define heuristics that match groups of syntactically similar element names. For each element name, identify element names that are syntactically similar to it. We rely on three methods to determine syntactic similarity: (i) common radical (using stemming); (ii) common suffix; and (iii) Levenshtein distance less than a given threshold (where the threshold can be adjusted). Each element and its syntactically similar elements are presented to the annotator as a group (starting with the most recurrent elements). Through inspection, the annotator decides which elements in the group are related and removes unrelated elements. The annotator then assigns annotation heuristics to each cleaned group. Note that this phase can be repeated. Indeed, while processing the top-50% most "promising" elements, the corresponding groups are cleaned, and in this process it may happen that some elements are dropped out and need to be revisited.
- *Create annotation heuristics for the rest of elements*. Continue visiting the remaining elements using the same principle (from most recurrent to less recurrent), again grouping elements by their syntactic distance so that the annotator can attach annotation

heuristics to groups of elements rather than individual elements.

During these phases, some elements are removed from a group and are left for processing in subsequent iterations. This happens during the manual inspection, if it is identified that one of the following applies:

- Documentation of elements with identical or syntactically similar names refer to meanings, that have no practically applicable common denominator (e.g. "table" is interpreted in some cases as a piece of furniture and in other cases as a database table);
- Neither has the element been documented nor do the parent elements provide an unambiguous context to understand the meaning of the element (e.g. an element occurs in a complex type defining a string array as a sequence of string elements, whereas the array is used to represent list of countries, company names, etc).

Annotation heuristics are represented as rules of the following form: *entity_reference* ← *synset* (e.g. *Password* ← *{password, pwd, strPassword, authpassword, pass}*). The meaning of such a rule is that an XML schema element matched by any element in the *synset* is mapped to the entity reference on the left-hand side of the rule. A synset, or a synonym ring, is a group of labels that are considered semantically equivalent. In the second and third phases of the methodology we use a variant of synsets in which the group of labels that are considered semantically equivalent is not fully enumerated, but instead it may be represented using regular expressions. Accordingly, we define a notion of *extended synset* – a set consisting of labels and regular expressions. A label matches a synset if it matches any of the elements in the synset. An example of an extended synset is *{*street, str*name, s_name}*. The labels "str. name", and "street_name" match this extended synset since they match the second element in the synset. In this context, the "*" stands for any sequence of characters (including the empty sequence). In defining such synsets with regular expressions, over-generalization may occur. For example, the regular expression '*str*name'* matches the string "string_name" which is not directly related to "street_name". This is natural since we are dealing with heuristic rules rather than exact rules.

Annotation heuristic rules may be partially overlapping. This simply means that a given element in a schema may be associated with multiple annotations. For example, a schema element <university_name> may represent both an educational institution name and a business name, and thus this element may have two (or more) semantic annotations.

In the proposed methodology, annotation heuristics are created as follows:

- Select a leaf node name;
- Inspect documentation of all lead nodes that have this name; if there is no documentation extracted from the XML schema or WSDL description, analyze the data structure, i.e. inspect the ancestors and siblings of each lead element.
- Based on documentation find a common, most specific semantic denominator(s) to attach to the element name. For example, "company name" and "person name" can be generalized in some domains as "customer name"):
  - If the annotator finds a common most specific semantic denominator for the element name, she creates a new annotation rule or extends an existing annotation rule that already covers the element name;
  - If no common denominator is found, the element name is skipped. In this case, finer-grained analysis is required and the element may be visited again at a later stage. For example, an element called "identifier" might refer to a business identifier or a customer identifier or a transaction identifier. So it might not be possible to define an annotation heuristics for this element. This will lead to certain element names not leading to any annotation heuristics and these element names might then remain un-annotated.
- When identifying the most specific common denominator in the previous step, the annotator should refer to the existing entities in the model (vocabulary, ontology or taxonomy). If there is no concept that matches the concept chosen as the common denominator, the model needs expansion. In this case, the annotator creates a new entity, or a creates a placeholder for a new entity and adds the details of this entity later. Although the methodology does not require a model *ex ante*, the existence of a model right from the start simplifies the annotation process. In case there is no predefined model, a list of entities is built manually during the annotation process.

III. Experimental results

In order to evaluate the applicability and usefulness of the proposed methodology we performed a case study using a repository of Estonian governmental Web services, namely X-Road [K+05]. The purpose of the case study is to analyze two aspects of the web services annotation methodology:

- To estimate the effort required to build ontologies and to semantically annotate existing web services;
- To estimate to what extent the required annotation effort can be reduced by applying the proposed methodology.

We analyzed 58 web service descriptions covering a total of 1045 operations. The input and output types of these operations are defined in XML Schema. Altogether, there are 7757 leaf elements in these schemas and ca 2900 distinct leaf node element names, meaning than on average each leaf

element name occurs 2.7 times in the repository. Manual annotation of such a number of schema elements would be a time-consuming project and thus specific methodologies and tools are required.

We applied the methodology for one iteration of phase 1 (annotation heuristics for most outstanding data elements), 4 iterations of phase 2 (annotation heuristics for most promising elements) and 1 iteration of phase 3 (annotation heuristics for syntactically grouped elements). Altogether we attached 5555 annotations to the 7757 leaf elements (70%). This was achieved after manually inspecting around 1600 leaf element names and creating almost 600 model entities in the process. We stopped the annotation process when more than two-thirds of the elements had been covered, after realizing that continuing the annotation process beyond that point was unlikely to yield additional insights.

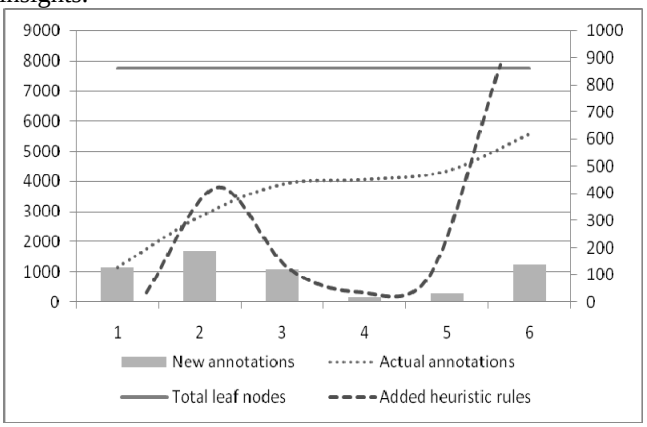

Figure 2. Annotation project characteristics.

Figure 2 shows the number of annotations achieved per iteration. Iterations are numbered 1-6 at the bottom of the figure, whereas 1 refers to iteration 1 in phase 1, numbers 2-5 refer to iterations 1-4 of phase 2, and number 6 refers to iteration 1 of phase 3. The numbers along the left-hand-side vertical axis indicate the number of annotations. The gray columns indicate the number of new annotations introduced during each iteration, while the dotted line indicates the total number of annotations achieved up to a given iteration (i.e. the sum of the number of annotations in the current and in all previous iterations). It can be seen that at the end of the sixth iteration, almost 6000 annotations are achieved. For reference, a horizontal line at the top of the figure indicates that the total number of elements being annotated is close to 8000.

The numbers on the right-hand-side vertical axis of the figure indicate the number of heuristic rules added per iteration. It can be observed that in the first iteration this number is low since of the elements selected in this phase occur very frequently, and therefore each annotation heuristic rule defined in this iteration leads to many annotated elements. The number then climbs sharply during the first iteration of the second phase where a large number of "promising" elements are annotated. The number drops in subsequent iterations of phase 2. Finally, the number climbs sharply again in the last iteration (of phase 3) since in this iteration, almost every element name needs its own annotation heuristics (around 900 annotation heuristic rules are added in this final iteration).

We conclude that phase 1 results in simple, yet powerful heuristics providing excellent *annotation per heuristic* ratio. Heuristic rules added in phase 2 provide annotations for elements with more domain-specific semantics, which naturally leads to a lower *annotation-per-heuristic* ratio. Finally, in phase 3 an order of magnitude more annotation heuristics need to be constructed since we are then dealing with very specific (generally one-off) elements and model entities.

Altogether 573 model entities were needed to annotate 5555 of 7757 possible XML Schema leaf node elements (see Table 1 for more information). Our empirical results show that the top 30 most recurrent entities provide semantic annotations to ca 15% of nodes, while the top 300 provides coverage to ca 50% of elements. The most important entities are listed in Table 2. Please note the differences between Table 1 and Table 2 – while in Table 1 XML Schema element names are listed, in Table 2 entity labels from the semantic model are listed.

Table 1. 12 most frequent element names in X-Road WSDL documents.

Element name (in Estonian)	**English interpretation**	**Occurrences**
isikukood	National identity code	205
teade	Notification	177
eesnimi	First name	144
perenimi	Surname (last name)	94
nimi	Name	87
kood	Code	77
synniaeg	Birth date	67
id	Identifier	66
liik	Type	45
et_nr	<does not translate>	44
kuupaev	Date	42
kpv	Date (abbreviated)	40

Table 2. Top 12 most frequently referred entities.

Entity label	**References**
National identity code	271
Date	203
General identifier	199
Notification	180
First name	178
Last name	165
Start date	162
Name	145
Birth date	132
End date	115
Address	85
Business registry code	72

In principle, every element name visited during an iteration of the methodology leads to an annotation heuristics that covers all occurrences of that element. However, as explained earlier, it sometimes happens that element names are ambiguous (e.g. "identifier") and no annotation heuristic can be defined that covers all of its occurrences. Therefore, the actual number of annotations produced after processing a given number of element names may be lower than the expected number of annotations. Figure 3 shows the divergence between the expected and the actual number of annotations produced by the heuristics. In this figure, the numbers along the horizontal axis are sequence numbers of element names, ordered by their occurrence frequency. The vertical axis gives the cumulative number of annotations (expected or actual) at a given point during the annotation process. The dotted and dashed lines in the figure show the estimated and the actual number of annotations respectively. The estimated number of annotations is calculated by assuming that for every element name, we would be able to define an annotation heuristic that produces annotations for all occurences of this element name. Figure 3 suggests that by processing the top 10% most recurrent element names (cf. first vertical dashed line) we would obtain annotation heuristics for ca. 50% of all elements, while processing the top-50% most recurring element names would cover ca. 80% of all elements (cf. second dashed line). However, the dashed line reveals that in practice, only ca. 40% of elements are covered after processing the top-10% most recurring element names, and ca. 55% of elements are covered after processing the top-50% most recurring element names. Still, the results show that the methodology is relatively effective when compared to manual, one-by-one annotation approach.

From a project management point of view, when allocating resources for annotation activities, there is a need to estimate, based on the characteristics of sources that need to be annotated, how many man-hours should be allocated. Since there is no cost estimation model for semantic annotation available, we publish the figures we observed in Figure 4. Like in software engineering, accurate effort estimation is a difficult problem, but we hope that these empirical results serve as initial reference point for future projects or studies on semantic annotation cost analysis. The empirical results in Figure 4 should be aligned with those in Figure 2, where 6 iterations took altogether 36 hours. In this light Figure 4 confirms that annotations of phase 1 and phase 2 were created, by using heuristic rules, much more effectively than those constructed in phase 3.

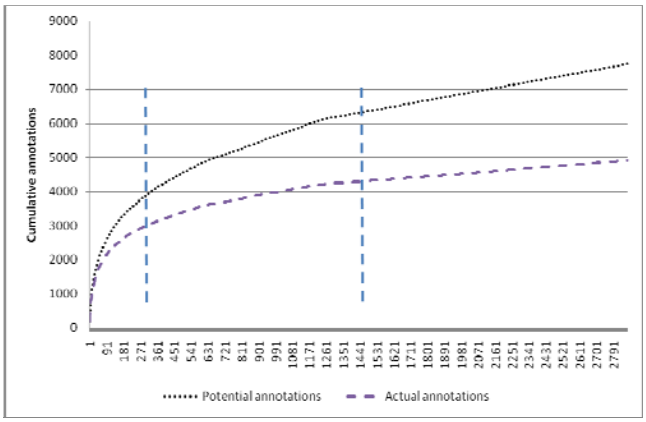

Figure 3. Comparison of potential and actual annotations constructed with heuristic rules.

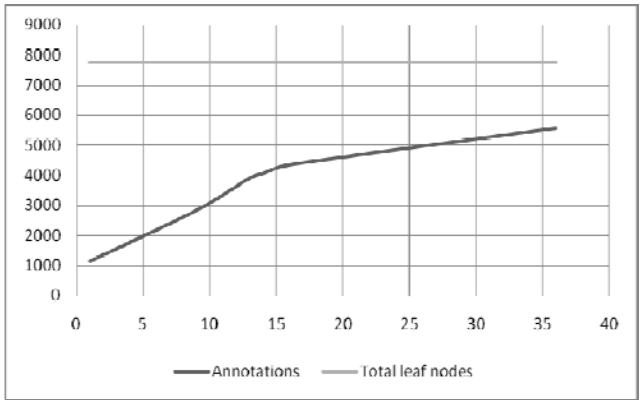

Figure 4. Man hours required for annotation.

Figure 5 shows the number of model entities created/used during the annotation (right-hand side vertical axis scale) and the number of annotations per entity (left-hand side scale). This figure reveals that proportionally, few entities are introduced at the beginning of the project but these entities are used to annotate many elements. This is a feature of our methodology, since the most outstanding enties are identified first and annotation heuristics are created for them. Subsequently, more and more annotations need to be introduced to cover the less recurrent elements, but eventually the number of model entities required stabilizes. It should be noted that while the number of new entities per annotation gradually decreases, the time required to provide new annotations gradually increases. This is shown in **Figure 6**, where the usefulness of annotation heuristics through different stages of the annotation process are presented.

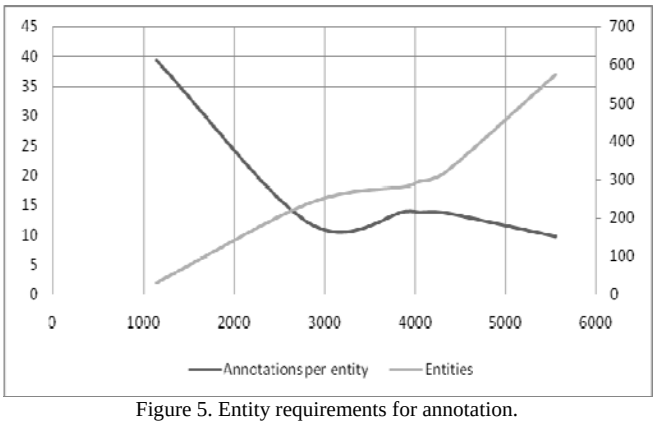

Figure 5. Entity requirements for annotation.

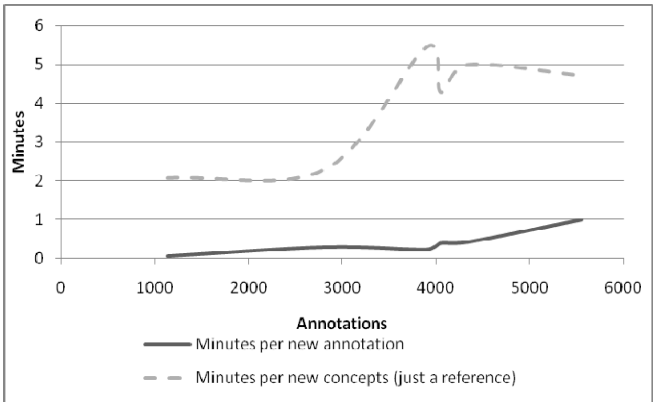

Figure 6. Efforts to add new entities and annotations.

IV. Related work

A significant amount of prior work has considered the automated annotation of Web content using shallow natural language processing (e.g. morphological analysis, named entity recognition) and disambiguation techniques [EMS+00, DEG+03]. Semi-automatic and automatic approaches to full-text semantic annotation have proved to be relatively scalable. Dill et al. [DEG+03] present a case study in which they apply such techniques to annotate a corpus of 264 million web pages. However, the precision of these techniques is limited by the fact that they deal with unstructured content. In the context of web service annotation, finer-grained techniques that exploit the structure of XML documents are applicable.

The METEOR-S semantic annotation framework [POS+04] applies shallow NLP techniques for semantic annotation of web service descriptions. The techniques employed in the METEOR-S framework derive from schema matching techniques: the problem of semantic annotation is seen as one of matching XML schema elements to entities in an ontology. This matching problem is addressed by combining shallow NLP techniques (tokenization, NGRAM, stemming, stop-word removal, etc.) with structural matching techniques. The ASSAM toolset [HJK04], on the other hand, applies machine learning techniques to classify web services, their operations and input and output message types. From this classification, a mapping is derived between a collection of web service descriptions and an ontology, using string similarity metrics.

The above techniques require pre-existing taxonomies or ontologies. This is not a suitable assumption in many settings. For example, governmental Web service repositories, such as the one considered in this paper, cover a wide diversity of domains including taxation, justice, education, public health, public infrastructure, etc. It is unrealistic to assume that a fixed ontology or combination of ontologies can provide the full extent of concepts required to semantically annotate such repositories. Moreover, some of the above techniques require linguistic knowledge bases that are not always available in all languages. Northern European or Baltic languages do not have linguistic knowledge bases as sophisticated as those available for English or German.

Burstein [B04] is concerned with constructing ontology mappings between terms used in different semantic web services. It is argued that since web service providers do not use a shared ontology for describing semantically their web services, automated ontology mapping is required. This work is complementary to ours. Indeed, our approach produces Web service descriptions that are annotated with respect to a vocabulary built on-the-fly during the annotation process. To achieve integration with other semantically annotated Web services repositories, ontology mappings are required.

Sabou [S04] proposes an automated method for Web service annotation using software API documentations. The idea is that if an API implements a web service, then the semantics of the API corresponds to the semantics of the web service. Thus, the API documentation (in natural language) can be used to semantically enrich Web service descriptions. This approach differs from ours in that we treat Web services as black-boxes, and we analyze only the Web service interfaces, not their implementations.

Other work [SWG+05] has sought to extract domain ontologies for textual content attached to web services, by leveraging shallow natural language processing techniques (e.g. part-of-speech tagging). This work is complementary to ours, insofar as our method can be optionally bootstrapped using a domain ontology.

Semantic interoperability in e-government services has been identified as one of the key aspects in public sector e-services [MK08]. Many member states of the European Union have ongoing semantic interoperability initiatives, e.g. German's Deutschland Online Italy's and the Finnish FinnONTO initiative [H+08]. There are also pan-European initiatives, e.g. SEMIC (SEMantic Interoperability Centre) and semanticGov [V+06]. However, current efforts in building up full-fledged semantic infrastructures in the e-Government context are hampered by the cost of semantically annotating large repositories of information

assets (including Web service interfaces) with a sufficient amount of precision. In this paper, we have used the Web service repository of the X-Road [K+05] platform as a dataset for validating our proposed methodology. The X-Road platform supports the technical interoperability of the components and registries of the Estonian state information system.

The application of semantically annotated Web services has been widely studied. For example, Pathak et al [PBH07] show that semantic annotations, of Web services interfaces provide a suitable basis for analysing substitutability of Web services. Meanwhile, Gomadan et al. [GVS+06] studied the application of semantic Web service annotations for Web service discovery, while Rao et al. [RKM06] and Pistore et al. [PST06] study their application to Web service composition. Finally, to estimate of the degree of human involvement in XML schema mediation Gomadam et al [GRR+08] introduce the concept of mediatability and provide a quantifiable and computable definition for it. Their experiments demonstrate that in average partial semantic annotations of schemas improve the mediatability by a factor of 2 while having complete annotations improves the mediatability by a factor of 3. These results indicate that many efforts can be saved in integration projects, if schemas are at least partially annotated.

V. Conclusions and future work

In this paper we propose a practical method for semantically annotating large collections of Web service descriptions. The method is based on syntactic analysis of XML schemas, which results in manual construction of annotation heuristics. While applying these heuristics Web services descriptions are annotated semantically.

We evaluate this method on a large repository of governmental Web services. The evaluation shows that some relatively simple techniques are quite cost-effective, saving hundreds or even thousands of man-hours of semantic annotation effort. The proposed method does not assume the availability of a pre-existing ontology or controlled vocabulary. Instead, the space of annotations is dynamically built during the annotation process.

Although we considered in this evaluation focuses on XML schema leaf node names, our approach can also be applied to annotate XPath expressions or XSD schema designators leading to either leaf nodes or internal nodes, as required. Furthermore, our experiments show (see Figure 4) that string similarity cannot be the sole criterion for label matching in of the context of Web services, if detailed distinction is required (such as to distinguish between “national identification code (NIC)” vs “NIC of company’s owner”). Moreover, there exist leaf nodes, whose names are identical, but the meaning is completely different (“code” could refer to a person’s identifier or company’s or a real estate object’s identifier or a postal code). Thus there is still need for defining finer-grained annotation heuristics, and here XPath expressions of XSD component designators could be used to refer to elements in a finer-grained manner.

We have implemented a tool that supports the proposed methodology. Our annotation toolset currently imports WSDL and XSD documents, applies a set of annotation heuristics, and finally exports the annotations as SA-WSDL model references injected within the imported documents. We aim to provide our annotation tool together with the constructed annotation heuristics for sample web service repositories as a public Web service such that semantic Web services research community would globally benefit from using it for providing semantics to their Web services descriptions.

Another thread of our future work includes applying the proposed methodology to common data interchange models encoded in XML Schema such as HL7, SID, ARTS, XBRL, XML-HR etc. In fact we have analyzed HL7 data model and our initial findings indicate that although unified data model annotation heuristics pointing to high-level entities are easier to construct (synsets are simpler) and annotations converge faster, more care should be taken on disambiguation due to homogeneous naming rules. The most commonly occurring names are for instance “id”, “code”, “typeid”, “templateid”, “realmcode” from altogether 345 different names. These names need to be disambiguated by analyzing the context in which they occur. This is another reason for investigating usage of XPath expressions in building annotation heuristics. For instance in HL7 the following 2 XPath locations for element named “addr” - *xsd:complexType[@name="COCT_MT030200UV04.Guardian"]/xsd:sequence/xsd:element[@name="addr"]* and *xsd:complexType[@name="COCT_MT030200UV04.Student"]/xsd:sequence/xsd:element[@name="addr"]* refer to two different entities – an address of a student and an address of a guardian though they both would refer to the same high-level entity representing an address.

Though common data models provide a standardized set of interfaces/data models, they are in practice not so trivial to apply due to their size and domain-specific nature. Moreover, in turns out that no common data model can completely cover the needs of all integration/development projects, they are used in. Thus, for the sake of maintainability, there is a need to migrate ad hoc data structures to standardized data models while at the same time extending the latter with application-specific parts. Our hypothesis is that annotation heuristics can help in migrating *ad hoc* data structures of existing integration projects into standardized data models.

Acknowledgment

This research is funded by the European Regional Development Fund through the Estonian Centre of Excellence in Computer Science.

REFERENCES

[B04] M. Burstein, “Ontology mapping for dynamic service invocation on the Semantic Web,” Proceedings of AAAI Spring Symposium on Semantic Web Services, Palo Alto, CA, USA, March 2004.

[BH08] M. Brian Blake and Michael N. Huhns, “Web-scale workflow: integrating distributed services,” IEEE Internet Computing, vol. 12, no. 1, 2008, pp. 55-59.

[DEG+03] Stephen Dill, Nadav Eiron, David Gibson, Daniel Gruhl, R. Guha, Anant Jhingran, Tapas Kanungo, Kevin S. McCurley, Sridhar Rajagopalan, Andrew Tomkins, John A. Tomlin, and Jason Y. Zien, “A case for automated large-scale semantic annotation,” Web Semantics: Science, Services and Agents on the World Wide Web, vol. 1, no. 1, December 2003, pp. 115-132.

[EMS+00] M. Erdmann, A. Maedche, H. Schnurr, and S. Staab, “From manual to semi-automatic semantic annotation: about ontology-based text annotation tools,” Proceedings of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content, Saarbrücken, Germany, August 2000.

[FL07] J. Farrell and H. Lausen, “Semantic Annotations for WSDL and XML Schema (SAWSDL),” W3C Recommendation, 2007, http://www.w3.org/TR/sawsdl.

[GRR+08] K. Gomadam, A. Ranabahu, L. Ramaswamy, A. P. Sheth, and K. Verma, "Mediatability: estimating the degree of human involvement in XML schema mediation," IEEE International Conference on Semantic Computing, Irvine, CA, USA, August 2008, pp. 394-401.

[H+08] E. Hyvönen, K. Viljanen, J. Tuominen, and K. Seppälä, “Building a national semantic web ontology and ontology service infrastructure - the FinnONTO approach,” Proceedings of the European Semantic Web Conference (ESWC), Tenerife, Spain, June 2008.

[HJK04] A. Heß, E. Johnston, and N. Kushmerick, “ASSAM: A tool for semi-automatically annotating semantic web services,” Proceedings of the 3rd International Semantic Web Conference (ISWC 2004), Hiroshima, Japan, 2004.

[K+05] A. Kalja, A. Reitsakas, and N. Saard, “eGovernment in Estonia: best practices,” Technology Management: A Unifying Discipline for Melting the Boundaries, IEEE, 2005, pp. 500-506.

[MK08] S. Muthaiyah and L. Kerschberg, “Achieving interoperability in e-government services with two modes of semantic bridging: SRS and SWRL,” J. Theor. Appl. Electron. Commer, vol. 3, no. 3, December 2008, pp. 52-63.

[NMN09] A. De Nicola, M. Missikoff, and R. Navigli, “A software engineering approach to ontology building,” Information Systems, vol. 34, no. 2, April 2009, pp. 258-275.

[NVS+06] M. Nagarajan, K. Verma, A.P. Sheth, J. Miller, and J. Lathem, “Semantic interoperability of web services - challenges and experiences,” Proceedings of the IEEE International Conference on Web Services (ICWS), Chicago IL, USA September 2006, pp. 373-382.

[ODM] “Ontology definition metamodel 1.0 beta, OMG adopted specification,” The Object Management Group (OMG), http://www.omg.org/technology/documents/modeling_spec_catalog.htm#ODM.

[OWL04] D. McGuinness and F. van Harmelen (editors), “OWL web ontology language, W3C Recommendation,” February 2004, http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[PBH07] J. Pathak, S. Basu, and V. Honavar, ”On context-specific substitutability of web services,” Proceedings of the IEEE International Conference on Web Services (ICWS), Salt Lake City, UT, USA, July 2007, pp. 192-199.

[POS+04] A. Patil, S.A. Oundhakar, A.P. Sheth, and K. Verma, ”Meteor-s web service annotation framework,” Proceedings of the International World Wide Web Conference (WWW), New York, NY, USA, May 2004, pp. 553-562.

[PST06] M. Pistore, L. Spalazzi, and P. Traverso, “A minimalist approach to semantic annotations for web processes compositions,” Proceedings of European Semantic Web Conference (ESWC), Budva, Montenegro, June 2006, pp. 620-634, 2006.

[RKM06] J. Rao, P. Küngas, and M. Matskin, “Composition of semantic web services using linear logic theorem proving,” Information Systems, vol. 31, nos. 4–5, pp. 340–360, 2006.

[S04] M. Sabou, “From software APIs to Web service ontologies: a semi-automatic extraction method,” Proceedings of the Third International Semantic Web Conference (ISWC2004), Hiroshima, Japan, November, 2004.

[SWG+05] M. Sabou, C. Wroe, C.A. Goble, and G. Mishne, “Learning domain ontologies for web service descriptions: an experiment in bioinformatics,” Proceedings of the International World Wide Web Conference (WWW), Chiba, Japan, May 2005, pp. 190-198.

[V+06] T. Vitvar, M. Kerrigan, A. van Overeem, V. Peristeras, and K. Tarabanis, “Infrastructure for the semantic pan-European e-government services,” Proceedings of the AAAI Spring Symposium on The Semantic Web meets E-Government (SWEG), Stanford University, CA, USA, March 2006.