University of Jordan

From the Selected Works of Dr. Mutaz M. Al-Debei

2011

A Semi-Automatic Ontology Extension Method for Semantic Web Services

Mutaz M. Al-Debei
Mohammad Mourhaf Al Asswad

Available at: https://works.bepress.com/mutaz_al-debei/16/
A Semi-Automatic Ontology Extension Method for Semantic Web Services

Mutaz M. Al-Debei¹, and Mohammad Mourhaf AL Asswad²
¹The University of Jordan, Department of Management Information Systems, Amman, Jordan
Email: M.AlDebei@ju.edu.jo
²Brunel University, Department of Information Systems and Computing, London, UK
Email: Mohammad.Al-Asswad@brunel.ac.uk

Abstract—this paper provides a novel semi-automatic ontology extension method for Semantic Web Services (SWS). This is significant since ontology extension methods those existing in literature mostly deal with semantic description of static Web resources such as text documents. Hence, there is a need for methods that can serve dynamic Web resources such as SWS. The developed method in this paper avoids redundancy and respects consistency so as to assure high quality of the resulting shared ontologies.

Index Terms—Ontology, Ontology Extension, Semantic Web Services, SWS, Name-based Matching, WSDL, Annotation

I. Introduction and Background

Ontologies are fundamental components of the Semantic Web as they are used to provide precise, explicit and shared meanings of Web resources. Ontology can be described as a definition of concepts, axioms and relations between concepts in a formal, shared and machine-understandable format [1; 2]. Ontologies have been applied to a wide range of computer applications such as knowledge engineering and sharing, database design, Artificial Intelligence and Web services. Semantic Web Services (SWS) are Web resources that provide more agile and efficient activities (i.e. discovery, composition and execution monitoring) of Web services than conventional syntactic ones. For SWS activities to be automatically performed by software agents and other Web services and applications, they need to be annotated to shared ontologies so as to provide precise, shared and semantically rich meanings to SWS elements. Semantic annotation of SWS involves referencing WSDL (Web Service Description Language) elements such as components of XML Schema to appropriate constructs of shared ontologies according to SAWSDL (Semantic Annotation for Web Service Description Language) notation. However, it is unlikely for shared ontologies to cover different knowledge domains comprehensively. To solve this problem, ontology engineers need either to build a more comprehensive ontology from scratch or to reuse and extend existing ones in order to provide meanings for Web service elements those having no correspondences on the shared ontology side. The first option (i.e. ontology building) can be done either manually or automatically. Manual ontology building is very difficult and labor-intensive task [3] as it requires ontology engineers to go through a long process which includes several phases [See 4]. Automatic ontology building cannot provide good quality ontologies due to the immaturity of existing learning methods [See 5]. Retrospectively, ontology extension has been proposed as an effective and efficient solution to many applications that use ontologies dynamically [6]. So, once an application is changed or new requirements are added, the ontology can be updated to accommodate new semantics for the changes. Ontology extension can be simply defined as the process of adding new ontological constructs to an existing ontology [7] while keeping the consistency of the ontology and avoiding any potential redundancy. These constructs can belong to any type of ontological entities such as classes and properties. Providing a novel and dynamic ontology extension method for SWS is the main aim of this paper. The rest of this paper is structured as follows. Next, we provide an overview about XML schema of WSDL and explain the annotation process. Then, we present the proposed ontology extension method. Thereafter, we provide a concise case through which we illustrate and evaluate the proposed method. Before providing the paper’s conclusions, we present some important related work and we show the contribution of this paper.

II. WSDL Structure and the Annotation Process

It is necessary to analyze the WSDL general structure first in order to make clear how WSDL elements can be semantically described. In overall terms, a WSDL file is composed of an element declaration, type definition, interface, binding and service. The element declaration, type definition and interface provide an abstract definition of a service, while binding and service describe the implementation aspects of a service [8]. Element declaration and type definitions are defined in the schema part of a WSDL file and provide data type definitions for input and output messages of operations and their parts. In an XSD, the elements that are direct children of a schema element are called global elements. Other XSD elements are called local elements. Furthermore, sub-elements of a complex type element are called direct child elements of that complex type element. To give more insights, Figure 1 presents a snapshot of a WSDL file of a Publication Information service. The binding and service elements of this service are removed due to space limitation. The data type definition (XSD) part of this WSDL document defines two global complex element; namely ‘Book’ and ‘JournalArticle’. The ‘Book’ element has six local child elements; namely ‘ISBN’, ‘BookTitle’, ‘Author’, ‘PubDate’, ‘PublishingOrganization’, and ‘VendorPrice’. These data types are used to define data of input and output message parts of WSDL operations. The ‘JournalArticle’ element, on the other hand, has also six local child elements; namely...

![Figure 1: A Snapshot of a Publication Information Service WSDL file.](image1)

Based on the previous brief analysis of WSDL elements, one can conclude that XSD elements including simple types (i.e. those having no child elements) in addition to complex types along with their child elements should be annotated since they describe data of operations’ messages. Other WSDL elements such as bindings and service define technical details and thus do not require semantic annotation. XSD definition embeds implicit semantic information that requires disambiguation, however. For example, the relation between a complex type and each of its child elements is similar to an ontological property. The annotation process can simply be described as finding correspondences in a shared ontology for Web service elements. The annotation process is either conducted manually [See 9]; or semi-automatically [10].

![Figure 2: Example of the Annotation Process.](image2)

Figure 2 represents an example of how elements of an Information publication service are annotated to a shared ontology describing the publication domain. According to the provided case in Figure 2, only five elements (one complex element; i.e. “Book” and four child elements; i.e. “ISBN”, “BookTitle”, “PublishingOrganization”, and “VendorPrice”) can be annotated to appropriate ontological elements (see Table 1). Therefore, the ontology should be extended in order to accommodate correspondences for the rest of elements included within the given WSDL file.

<table>
<thead>
<tr>
<th>Web Service Elements</th>
<th>Ontology Constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book</td>
<td>Book</td>
</tr>
<tr>
<td>BookTitle</td>
<td>Title</td>
</tr>
<tr>
<td>PublishingOrganization</td>
<td>Publisher</td>
</tr>
<tr>
<td>VendorPrice</td>
<td>Price</td>
</tr>
</tbody>
</table>

In the next section, the proposed ontology extension method is explained.

### III. The Proposed Extension Method

Ontologies used for annotation should be extended when they do not have correspondences for service concepts. In the context of this paper, an ontology is extended by adding a class and/or object property. When extending an ontology, a class should not be added to the ontology as an isolated entity. This is because ontological classes normally participate in relations (properties) that form axioms. Object properties can be seen as links that relate two or more classes together. Subsequently, a newly added class should be appropriately linked to other classes in an ontology. The extension method proposed in this paper defines two extension types; namely complex and child elements extension. Complex elements have a main concept and a set of child elements that are conceptually linked to the main concept with implicit relations that are derived from the XSD structure of complex types. The ontology extension method presented in this paper utilizes OWL API [11] to extract and add ontological constructs to shared ontologies. The OWL API is a Java API that allows developers to manipulate ontologies represented in the Web Ontology Language (OWL) formalism. In addition, CN-Match algorithm is used in this paper to perform the required matching of ontological constructs. CN-Match [see 12] is an automatic name-based matching algorithm that enables effective and precise matching between single terms and compound nouns as it is (1) founded on linguistics theories; (2) exploit external thesauri such as WordNet; and (3) utilize Levenshtein Distance to measure similarities between strings [13]. The ontology extension method to accommodate missing correspondences to Web service child elements is demonstrated in Figure 3 and explained as follows.

1. Use CN-Match to check if the given ontology has a correspondence for the given child element (denoted in Figure 3 as CorCE). This process is important since it prevents any potential redundancy that may result from adding extra construct that denotes the same meaning as the meaning of the given child element. The required correspondence of the given child element could exist in the ontology but might not be linked through an object property to the correspondence of the complex element (CorCoE). Consequently, it is very important that all classes of the given ontology are matched against missing child elements.

2. But, if the ontology does not contain such a
correspondence, create a class (Cls) that denotes the given child element. Next, create an object property and label it appropriately. The created class and the correspondence of the complex element are the domain and range of the new object property.

Using the case illustrated in section II, one can note that one complex element (i.e., 'JournalArticle') and three child elements (i.e., 'Author', 'PubDate', and 'Journal Name') cannot be annotated to appropriate constructs in the shared ontology. Hence, an extension to the shared ontology is required. As a further explanation about the application of the proposed extension method, we now show how 'Author' as a child element of the 'Book' complex element can be added to the existing shared ontology. In this case, CN-Match is used to check for a correspondence of 'Author' at the ontology side. CN-Match is unable to find any semantically appropriate correspondence. Therefore, a class called 'Author' and an object property called 'isWrittenBy' are added to the shared ontology. The domain of this object property is 'Book' and the range is 'Author'. In a similar manner, the other child element of 'Book' (i.e., 'PubDate') is added to the shared ontology.

On the other hand, extension for complex elements is performed as presented in Figure 4 and explained below.
1. Use CN-Match to check if the shared ontology has a correspondence for the given complex element (denoted in Figure 4 as CorCoE). Redundancy is also checked as in the extension for child elements.
2. If the ontology does not include such a correspondence, then add a class with appropriate label; otherwise go to step 3.
3. For every element included within the set of child elements of the given complex element, perform the extension method for child elements as illustrated in Figure 3.

To further explain the previous method, we also use the case presented in Section II so as to add the required correspondences. The WSDL file presented in Figure 1 shows that 'JournalArticle' is a complex element that seems to not having any correspondence in the shared ontology. Following the extension method for complex elements, CN-Match is performed and no appropriate correspondence for 'JournalArticle' is found. Therefore, a class called 'JournalArticle' is added to the shared ontology. According to the WSDL file, 'JournalArticle' has six child elements (i.e., 'JournalName', 'Author', 'ISBN', 'Publisher', 'ArticleTitle', and 'Price'). Each one of them is treated using extension method for child elements. By applying this extension method, CN-Match finds five appropriate correspondences ('ISBN', 'Title', 'Publisher', 'Price', and 'Author') for five child elements ('ISBN', 'ArticleTitle', 'Publisher', 'Price', and 'Author') of the 'JournalArticle' complex elements. Accordingly, five object properties (i.e., 'isIdentifiedBy', 'hasTitle', 'isPublishedBy', 'isPricedAt', and 'isWrittenBy') are added to the shared ontology to link 'JournalArticle' class to the five found corresponding classes. The domain of these five object properties is 'JournalArticle', while the ranges are the aforementioned five corresponding classes respectively. The 'JournalName' child element of the 'JournalArticle' complex element is treated differently as CN-Match is unable to find any appropriate correspondence for it. Therefore, a class called 'JournalName' and an object property called 'isPublishedIn' are added to the shared ontology. The domain of this object property is 'JournalArticle', while the range is 'JournalName'.

Having extended the shared ontology (see Figure 5) to accommodate correspondences for all elements included in the WSDL file of the publication information service, a complete annotation of this WSDL file is achieved as presented in Table 2.
The nature of provided data and the way in which this data can be used for extension. The proposed extension approach is different from the other approaches those existing in the literature in that: (1) It adds not only classes but also object properties to extended ontologies; and (2) it performs an important check using CN-Match to find out if a similar concept to the candidate one exists in the ontology. If such a concept exists, then it will be used instead of adding a new class. This later check process prevents any potential redundancies in extended ontologies. The ontology extension method developed in this research can extend ontologies by adding classes and object properties only. Other important ontological components such as datatype properties and sub-super class relationships can also be added during extension. Adding these extra structures could, however, complicate and delay the annotation and extension activities and thus the feasibility of this addition should be studied carefully in real, practical and industrial settings.

IV. Related Work and Contributions of this Paper

There are few approaches that aim to extend ontologies in an automatic or semi-automatic manner. For example, [14] provide an ontology extension method to add concepts and relations extracted from textual documents using NLP techniques. A different ontology extension approach is proposed by [6]. This approach expands ontologies semi-automatically by mining textual data of Websites. In the later approach, Spreading Activation, which is a semantic network search method, is used to find the most relevant terms to the given domain. These terms are then incorporated into the original ontology [6]. However, the aforementioned approaches are for static resources (i.e. text documents) and thus cannot be used for applications that require dynamic ontology extension. Moreover, redundancy check is not provided within these approaches and thus the quality of extended ontologies cannot be assured. On the other hand, the proposed extension method deals with dynamic resources (i.e. SWS). To the best of authors’ knowledge, this proposed method is the first to be implemented in order to deal with the important SWS in nowadays digital business. Moreover, the proposed method importantly avoids redundancy and ensures consistency while extending shared ontologies. This is because this method does not add any new ontological construct without performing redundancy check by utilizing CN-Match algorithm. Consistency of shared ontologies is assured by only adding constructs those can be linked to other constructs within the ontology for the sake of ontology commitment. In other words, concepts that do not fall within the scope of the shared ontology (i.e. isolated concepts) cannot be added. These two features (i.e. avoiding redundancy and respecting consistency) would result in high quality shared ontologies.

V. Conclusions and Future Work

The provision of a method for semi-automatic ontology extension is a contribution to the existing body of knowledge. The provided method can be used not only for Web service annotation but also for other applications such as annotation of HTML pages, semantic query answering and knowledge-based systems. The ontology extension idea is very new and thus few extension approaches exist in the ontology literature. Most existing approaches extend ontologies used for semantic description of textual and static Web resources. Consequently, there is a need for ontology extension methods that can serve dynamic Web resources. This is because dynamic resources are different from static ones in terms of the nature of provided data and the way in which this data can be used for extension. The proposed extension approach is different from the other approaches those existing in the literature in that: (1) It adds not only classes but also object properties to extended ontologies; and (2) it performs an important check using CN-Match to find out if a similar concept to the candidate one exists in the ontology. If such a concept exists, then it will be used instead of adding a new class. This later check process prevents any potential redundancies in extended ontologies. The ontology extension method developed in this research can extend ontologies by adding classes and object properties only. Other important ontological components such as datatype properties and sub-super class relationships can also be added during extension. Adding these extra structures could, however, complicate and delay the annotation and extension activities and thus the feasibility of this addition should be studied carefully in real, practical and industrial settings.

REFERENCES