Assigning Semantic Meanings to XML

Peishen Qi, Drew McDermott and Dejing Dou

Yale Computer Science Department
New Haven, CT 06520, USA
{peishen.qi,drew.mcdermott,dejing.dou}@yale.edu

Abstract. The Semantic Web Project of W3C is developed at the purpose of extending the current web to make the information automatically processable by software programs. RDF/RDFS, DAML+OIL and the recent OWL are several specifications proposed along this way. However, the proliferation of XML on the current web makes it a tempting target if we can make use of these existing resources. XML is good at marking up the structural relationships, but it has limited capability in presenting the semantics contained in the data.
In this paper, we describe our approach of extracting the propositional contents from XML documents and representing them in the ontology formalism. For any XML document, we first build up a *surface* ontology directly from its DTD file and encode the data in this ontology. We then represent the dataset in some more *standard* domain ontology via ontology translation. The *Meaning Definition Axioms* bridging over the surface ontology and the deep standard ontology have the function of assigning meanings to XML tags. The reverse direction which serializes a dataset of the standard ontology to a specific XML language works in the similar way. Combined with our former work, the whole system is capable of translating between the structural XML and the semantical RDF/DAML/OWL. It also can translate between heterogeneous XML documents about identical or similar domains of discourse.

1 Introduction

The information on the internet is growing at an exploding speed. The huge volume of resources makes it almost impossible to process them manually. The Semantic Web Project of the World Wide Web Consortium (W3C) aims at supplementing existing web content with formalized data or metadata, so that automated agents can process the data automatically [14]. In the Semantic Web model, RDF[1]/RDFS[2] use *metadata* to describe Web resources, and provide a standard for creating machine processable documents. DAML+OIL[3] and the recent OWL[5] extend RDF/RDFS with richer modelling primitives and work on the ontology level. However, automated agents can not step outside the RDF world to access the information in the XML documents that do not follow the RDF specification. These XML documents make a significant component of the current Web. Being able to make use of these existing resources would give agents access to a large pool of information without having to rewrite any documents.

We want to extract the propositional contents from XML documents, even though most XML specifications were not invented with that target in mind. In the other direction, a dataset of some ontology can also be serialized to a given XML specification, and become usable by more real-world applications implemented for that XML specification. The whole system is thus capable of translating between XML and RDF/DAML, and even among heterogeneous XML languages about identical or similar domains of discourse.

In our approach of assigning meanings to XML, a tool helps us first build up a *surface* ontology from the Document Type Definition (DTD) file associated with the XML document. A dataset using the vocabulary of the surface ontology then can be extracted from or serialized to the XML format. However, the surface ontology always mixes syntax with domain knowledge. In order to get rid of the syntax stuff, we need a *standard* ontology which only focuses on describing domain knowledge. Data are exchanged between the surface ontology and the standard ontology via our approach of ontology translation in [16]. More specifically, we will do ontology translation with the help of a set of logical formulas, which we call *Meaning Definition Axioms.* Fig.1 gives an overview of the whole structure, where *OntoEngine* [10] is our inference engine for ontology translation. The surface ontology is built up from the DTD file automatically. Extracting the propositional contents from an XML document to the dataset in the surface ontology can also be fully automated. The meaning definition axioms assign semantic meanings to XML languages and have to be written with the help of domain experts, thus involve human work. Once we have the MDAs, translation between the surface ontology and the standard ontology can be done automatically by OntoEngine.

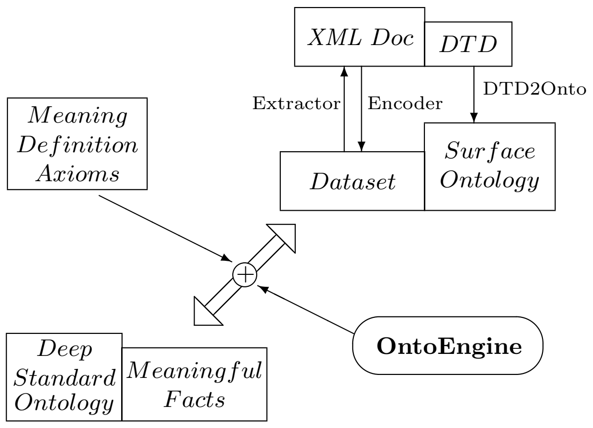

Fig. 1. translating between XML and RDF

We will use Lisp-like notation for those logical formulas, here Web-PDDL [15], a typed logic extension of PDDL [17] for web applications. It will also be our internal representation of ontologies and datasets. The formula traditionally written as $\forall x(P(x) \supset Q(x))$ will be written as `(forall (x) (if (P x) (Q x)))`. If type declarations are involved, we declare the types of variables by writing "—*vars*— - *type*," as in

```
(forall (x y - @Animal:animal)
    (if (and (@Animal:predator x) (bigger x y))
        (@Emotion:fears y x)))
```

where the prefix before each term introduced by the @ sign declares the namespace where that term comes from. Symbols without a prefix are of the local namespace.

2 Meanings Conveyed by XML Documents

XML can be seen as a data-centric language. The following XML document describes a job position seeker [13], whose name is "Jeff Seeker".

```
<JobPositionSeeker status="active">
  <PersonalData>
     <PersonName>
        <FormattedName>Jeff Seeker</FormattedName>
     </PersonName>
     <PostalAddress>
       <PostalCode>06511</PostalCode>
     </PostalAddress>
  </PersonalData>
  <Resume>...</Resume>
</JobPositionSeeker>
```

We focus on three kinds of meanings [12] conveyed by XML documents:

1. Objects in classes, such as a job position seeker, or a Resume. They correspond to instances of DAML+OIL *Class*es;
2. Attributes of the objects, such as the name of the job position seeker. The similar concept in DAML+OIL is the *DataTypeProperty*;
3. Associations between objects, for example, a Resume gives more detailed information (education, work experience, etc.) of a job position seeker. In DAML+OIL, it is the *ObjectProperty* that captures these relationships.

These kinds of information can be modelled more explicitly in RDF/RDFS, DAML+OIL or OWL using RDF triples. Fig. 2 shows the arc diagram for an RDF triple <*subject predicate object*>, which means *subject HAS predicate object.* The following set of RDF triples denotes the meanings contained in the above XML document.

<JobPositionSeeker_111 *status* active>
<JobPositionSeeker_111 *name* "JEFF SEEKER">

<JobPositionSeeker_111 *address* PostalAddress_200>
<PostalAddress_200 *zipcode* 06511>
<Resume_1 *describe* JobPositionSeeker_111>

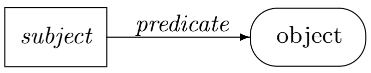

Fig. 2. The arc diagram for an RDF triple $< subject, predicate, object >$

In the subsequent sections, we will describe our method of extracting the propositional contents from XML documents and representing them in the ontology formalism. The opposite direction of translation is also interesting and needed since embedding datasets in XML format can make them usable in much wider real-world applications.

3 Build up the Surface Ontology from a DTD File

We assume that there exists a Document Type Definition (DTD) file for each XML document. When the DTD file is absent, automatical tools can help construct it from the XML document. Our idea should also work for XML Schema. The following is a simplified version of the DTD file for the above JobPositionSeeker [13] XML document. Its full version can be accessed at [6].

```
JobPositionSeeker.dtd
<!ELEMENT JobPositionSeeker (PersonalData, Resume)>
<!ATTLIST JobPositionSeeker status (active | inactive) #IMPLIED>

<!ELEMENT PersonalData (PersonName, PostalAddress)>
<!ELEMENT PersonName (FormattedName, GivenName, FamilyName, ...)>
<!ELEMENT FormattedName (#PCDATA)>
<!ELEMENT PostalAddress (PostalCode, ...)>
<!ELEMENT PostalCode (#PCDATA)>

<!ELEMENT Resume ...>
```

An XML DTD file defines the legal syntactic building blocks of an XML document. It can give the human readers clues about the default semantics of the XML language through the document structure and the text of tags. The first step in our approach is to build up a *surface* ontology from the DTD file. The surface ontology captures the default semantics and acts as one end in the following ontology translation process.

Building up the surface ontology is quite straightforward and can be fully automated. Currently we only focus on the ELEMENT and ATTRLIST keywords in a DTD file. Each ELEMENT entry defines a class. We create a Web-PDDL *type* for its name parameter, which is the first parameter after the keyword. Furthermore, a *predicate* is created to link this type to each of those types corresponding to the remaining parameters, respectively. The above DTD file has seven ELEMENT entries, so we would create seven *type*s for them. The first entry says that the JobPositionSeeker element must contain the PersonalData, followed by a Resume. Thus we need to introduce three *predicate*s which link JobPositionSeeker with the other three. The ATTRLIST entries define attributes of classes. They are also represented as Web-PDDL's binary *predicate*s, but with the second arguments being built-in types (such as *String*), or those domain specific types composed of enumerated instances. The "status" attribute of JobPositionSeeker is an example of the latter.

The surface ontology in Web-PDDL for the above JobPositionSeeker DTD looks like:

```
(define (domain jobpositionseeker-ont)
   (:types
        JobPositionSeeker - Object
        PersonalData Resume - Object
        PersonName PostalAddress - Object
        FormattedName GivenName FamilyName -String
        PostalCode - String
        JobPositionSeeker_status - Object
   )
   (:constants
        active - JobPositionSeeker_status
        inactive - JobPositionSeeker_status
   )
   (:predicates
       (hasPersonalData jo1 - JobPositionSeeker pe2 - PersonalData)
       (hasResume jo1 - JobPositionSeeker re2 - Resume)
       (status jo1 - JobPositionSeeker st2 - JobPositionSeeker_status)
       (hasPersonName pe1 - PersonalData pn2 - PersonName)
       (hasPostalAddress pe1 - PersonalData po2 - PostalAddress)
       (hasFormattedName pn1 - PersonName fn2 - FormattedName)
       (hasGivenName pn1 - PersonName gn2 - GivenName)
       (hasFamilyName pn1 - PersonName fn2 - FamilyName)
       (hasPostalCode pa1 - PostalAddress pc2 - PostalCode)
   )
)
```

With the surface ontology at hand, information in the XML document can be automatically extracted out and represented as a set of logical facts using the vocabulary of the surface ontology. The dataset for the JobPositionSeeker XML document is given here:

```
(define (dataset jobpositionseeker-data)
   (:domain
      (uri "http://www.xml.org/xml/schema/7083341b/
            JobPositionSeeker-1_1.dtd" :prefix JobPos))
   (:objects
      JobPositionSeeker1 - @JobPos:JobPositionSeeker
      PersonalData3 - @JobPos:PersonalData
      PersonName4 - @JobPos:PersonName
      PostalAddress5 - @JobPos:PostalAddress
      Resume6 - @JobPos:Resume
   )
   (:facts
      (@JobPos:status JobPositionSeeker1 @JobPos:active)
      (@JobPos:hasPersonalData JobPositionSeeker1 PersonalData3)
      (@JobPos:hasPersonName PersonalData3 PersonaName4)
      (@JobPos:hasFormattedName PersonaName4 "Jeff Seeker")
      (@JobPos:hasPostalAddress PersonalData3 PostalAddress5)
      (@JobPos:hasPostalCode PostalAddress5 "06511")
      (@JobPos:hasResume JobPositionSeeker1 Resume6)
   )
)
```

One advantage of introducing the surface ontology is that itself and the dataset written in it both can be drawn from the DTD file and the XML document automatically. The surface ontology still maintains the tree structure of the original XML specification, thus datasets in it can be written back to the serialized XML format without much difficulty. Furthermore, as we will see, the surface ontology shares its first order logic syntax with the standard ontology. This makes it easier to build and understand the translation rules, both of whose sides now can have the same syntax.

Besides these, such kind of preprocessing can also alleviate future work in ontology translation. For example, in some XML documents using the JobPositionSeeker specification, address information could also appear somewhere for an institution or a company. The string "CT" might show up more than once and all denote the same meaning as the state of Connecticut. Our meaning extractor could detect that the different occurrences of "CT" were all tagged by Region and should correspond to objects of type *Region* in the surface ontology. Since these objects all have the same value, only one is actually created. This kind of identification in advance can save later work concerning equality judgement.

4 Translation between Surface and Standard Ontologies: the Meaning Definition Axioms

The transform described in Section 3 does not by itself yield a satisfactory translation of an XML document. The surface ontology can be built from the DTD file automatically. But in the formal sense, it is actually not an ontology. The surface ontology usually mixes up semantic and syntactic stuff, so the whole structure looks somewhat odd from the view of a domain expert. For example, in the domain describing personal information, many ontologies break a person's full name down into useful components such as family name and given name. So why do we need an extra FormattedName if we already have the GivenName and FamilyName? The above JobPostionSeeker surface ontology also uses a PersonalName type, which is meaningless and solely introduced because that's the way the XML tree was organized.

We need an ontology which concerns itself entirely with domain types and semantic relationships. We will call it a *standard* ontology. The standard ontology focuses on domain knowledge and is thus independent of the original XML specifications. Because its role in our ontology translation is contrary to the surface ontology, we also call it a *deep* ontology. A deep ontology for a domain might already exist, for example among the DAML ontology library [7]. Otherwise, a new one has to be built with the help from domain experts.

Once a standard ontology is selected, the remaining and more important step is to bridge the gap between the surface ontology and the deep standard ontology. We take this as an ontology translation problem. In [16], we have stated that ontology translation might be best thought of in terms of ontology merging. The merge of two related ontologies is obtained by taking the union of the terms, the axioms defining them, and new axioms bridging across the two ontologies. In this paper, the bridging axioms have the function of assigning meanings to XML documents. We give them a more specific name, *Meaning Definition Axioms.* Since only human can understand some complicated semantic relationships, building up these meaning definition axioms can not be fully automated. It must involve human's, especially domain experts' participation.

OntoEngine accepts a merged ontology, plus a dataset in one of its components (the *source ontology*), then outputs the dataset in another component (the *target ontology*). This pattern holds whether the source is the surface ontology and the target is the deep ontology, or vice versa. The same set of axioms can drive the translation in either direction. The inference pattern looks like: source $\Rightarrow$ target. All the axioms should have the format of $P_1 \wedge \cdots \wedge P_n \Rightarrow Q_1 \wedge \cdots \wedge Q_m$, where P_i's are predicates from either source or target ontology and Q_i's are predicates that only appear in target ontology. These inference rules can draw conclusions out of existing facts from both source and target vocabularies, and project onto target vocabulary only. If an axiom can work in both directions, we will use a $\Leftrightarrow$ (iff) to denote it.

We will give some examples of the meaning definition axioms below and explain how they work. On the XML side, we still stick to the JobPositionSeeker specification. We want to extract the meanings concerning Resume information from it and have found a DAML ontology about Resume on the web [9], which will act as the deep standard ontology. Our translator *PDDAML* [11] can automatically translate a DAML ontology to its Web-PDDL version. The following is an abridged version of the deep ontology focusing on the contact and education contents.

```
(define (domain ResumeOnt)
    (:types
        MyResume - Object
        ContactInfo Education - MyResume)
    (:predicates
        (contactName c - ContactInfo s - String)
        (contactAddress c - ContactInfo s - String)
        (contactTel c - ContactInfo s - String)
        (schoolName e - Education s - String)
        (schoolCity e - Education s - String)
        (schoolState e - Education s - String)))
```

In the JobPositionSeeker DTD, all the data are structured around the job position seeker. A job position seeker has his/her personal data, which include the information about name, postal address and telephone number, etc. The job position seeker also has a Resume for job application. In the Resume domain, the central information should be the Resume. The axioms below extract the personal data of a job position seeker and map them to the contact information of Resume in the deep ontology. The prefix *@JobPos:* denotes the surface ontology and *@Resume:* the deep ontology. Using prefixes, we can avoid name conflicts when handling terms from different namespaces.

```
(T-> @JobPos:PersonalData @Resume:ContactInfo)

(forall (pd - @JobPos:PersonalData
         fn - @JobPos:FormattedName)
        (iff (@Resume:contactName pd fn)
             (exists (pn - @JobPos:PersonName)
                     (and (@JobPos:hasPersonName pd pn)
                          (@JobPos:hasFormattedName pn fn)))))


(forall (pd - @JobPos:PersonalData
         tn - @JobPos:TelNumber)
        (iff (@Resume:contactTel pd tn)
             (exists (vn - @JobPos:VoiceNumber)
                     (and (@JobPos:hasVoiceNumber pd vn)
                          (@JobPos:hasTelNumber vn tn)))))

(forall (pd - @JobPos:PersonalData
         pa - @JobPos:PostalAddress
         cc - @JobPos:CountryCode
         pc - @JobPos:PostalCode
         r - @JobPos:Region
         m - @JobPos:Municipality
         d - @JobPos:DeliveryAddress
         al - @JobPos:AddressLine
         s - String)
        (if (and (@JobPos:hasPostalAddress pd pa)
                 (@JobPos:hasCountryCode pa cc)
                 (@JobPos:hasPostalCode pa pc)
                 (@JobPos:hasRegion pa r)
                 (@JobPos:hasMunicipality pa m)
                 (@JobPos:hasDeliveryAddress pa d)
                 (@JobPos:hasAddressLine d al)
                 (eval (@built-in:concatenate al "\n"
                          m ", " r " " pc "\n" cc) s))
            (@Resume:contactAddress pd s)))
```

The first axiom is an abbreviation saying that we give a one-to-one map between the two types: *@JobPos:PersonalData* and *@Resume:ContactInfo*. The other three axioms handle the name, address and phone number information, respectively. The *@built-in:concatenate* in the fourth axiom is a built-in data processing function in our OntoEngine, which concatenate its arguments (strings) into one whole string. Some other such functions include *@built-in:stringnumber* which converts between the string format and digit format of numbers.

Using these axioms, the meanings embedded in the following XML document

```
<JobPositionSeeker>
  <JobPositionSeekerId>foo-Y-CS-2000</JobPositionSeekerId>
  <PersonalData>
    <PersonName>
      <FormattedName>Jeff Q. Seeker</FormattedName>
    </PersonName>
    <PostalAddress>
      <CountryCode>US</CountryCode>
      <PostalCode>06511</PostalCode>
      <Region>CT</Region>
      <Municipality>New Haven</Municipality>
      <DeliveryAddress>
        <AddressLine>1701 Whitney, Apt.3</AddressLine>
      </DeliveryAddress>
    </PostalAddress>
    <VoiceNumber>
      <TelNumber>(203) 400-8000</TelNumber>
    </VoiceNumber>
  </PersonalData>
```

can be stated as a set of logical facts:

```
(:objects
   PersonalData3 - @Resume:ContactInfo)
(:translated-facts
   (@Resume:contactName PersonalData3 "Jeff Q. Seeker")
   (@Resume:contactAddress PersonalData3
            "1701 Whitney, Apt.3
             New Haven, CT 06511
             US")
   (@Resume:contactTel PersonalData3 "(203) 400-8000"))
```

which can further be written out in DAML using PDDAML, and become usable by Semantic Web agents.

```
<rdf:RDF
   xmlns:rdf   = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs  = "http://www.w3.org/2000/01/rdf-schema#"
   xmlns:daml  = "http://www.daml.org/2001/03/daml+oil#"
   xmlns:Resume= "http://orl01.drc.com/daml/Ontology/Resume/3.1/
                  Resume-ont.daml#">
 <rdf:Description rdf:about="">
   <daml:imports rdf:resource="http://orl01.drc.com/daml/Ontology/
                  Resume/3.1/Resume-ont.daml#"/>
 </rdf:Description>
 <Resume:ContactInfo rdf:ID="PersonalData3">
  <Resume:contactName>Jeff Q. Seeker</Resume:contactName>
  <Resume:contactAddress>
  1701 Whitney, Apt.3
  New Haven, CT 06511
  US
  </Resume:contactAddress>
  <Resume:contactTel>(203) 400-8000</Resume:contactTel>
 </Resume:ContactInfo>
</rdf:RDF>
```

Note that only those XML tags which are assigned meanings through the meaning definition axioms can induce useful information in the output dataset. Other stuff in the original XML document is ignored (e.g. the JobPositionSeekerId tag). As the understanding grows, we can improve the axioms and extract as much information as possible.

Up to now, we have been mainly dealing with the direction from the surface ontology to the deep ontology. Our meaning definition axioms should also help the translation in the other direction, which is needed when we want to serialize a set of XML-independent logical facts to some specific XML language. Several new problems will pop up. In the above axioms, the first three use "iff" and can work fine in both directions. The fourth axiom concatenate the different parts of a postal address together. More axioms are needed to translating postal address information back to XML, that is splitting an entire address string into separate parts. Due to the space limit, here we just give one which extracts out the post code part. With the help of some data sources such as the DAML Country Codes [8], the fetchPostalCode function first tests whether the country code is absent or present and thus determines the line (the last or the second to last one) on which the postal code should appear. Then it fetches out the string which matches the pattern "$[0 \sim 9]^+(-[0 \sim 9]^+)?$" from that line.

```
(forall (pd - @JobPos:PersonalData
         s pc - String)
        (if (and (@Resume:contactAddress pd s)
                 (eval (fetchPostalCode s) pc))
            (exists (pa - @JobPos:PostalAddress)
               (and (@JobPos:hasPostalAddress pd pa)
                    (@JobPos:hasPostalCode pa pc)))))
```

Another problem is that the dataset in the deep standard ontology only contains meaningful information. The above Resume dataset says nothing about the job position seeker. But in the surface ontology, an object of type *JobPositionSeeker* must exist so that we can have the root node for the tree structure of the XML document. Since we know that one XML document can have only one root element, we can add a constant `@JobPos:JobPositionSeeker1` into the merged ontology. The following axiom links the root to one of its sub-elements: PersonalData.

```
(forall (c - @Resume:ContactInfo)
        (@JobPos:hasPersonalData @JobPos:JobPositionSeeker1 c)))
```

With this new constant added, the reverse translation results in almost the same input XML document. The only thing missed is the JobPositionSeeker-ID element because there is no its correspondence in the DAML dataset.

One more problem that we need to take care of when writing the meaning definition axioms can be illustrated more clearly by the following example concerning the education information in the JobPositionSeeker document.

```
(T-> @JobPos:SchoolOrInstitution @Resume:Education)

(forall (si - @JobPos:SchoolOrInstitution
         re - @JobPos:Region)
    (iff (exists (ls - @JobPos:LocationSummary)
             (and (@JobPos:hasLocationSummary si ls)
                  (@JobPos:hasRegion ls re))))
         (@Resume:schoolState si re))

(forall (si - @JobPos:SchoolOrInstitution
         m - @JobPos:Municipality)
    (iff (exists (ls - @JobPos:LocationSummary)
             (and (@JobPos:hasLocationSummary si ls)
                (@JobPos:hasMunicipality ls m))))
         (@Resume:schoolCity si m))
```

The last two axioms both have an existential quantifier in them. During the translation, we will remove the existential quantifiers by skolemization [19]. More specifically, if we know that (`@Resume:schoolState Institution_1 "CT"`) holds, a skolem constant (e.g. LocationSummary_2) of type *@JobPos:LocationSummary* will be created and two new facts concerning it will be added to the target dataset. However, if we also have the fact (`@Resume:schoolCity Institution_1 "New Haven"`) in the source dataset, another skolem constant of *@JobPos:LocationSummary* can and will be created for the same `Institution_1`, because the two axioms have different universal quantifiers. This would finally result in an XML document with the state and city information of the same institution tagged in different LocationSummary tags. The problem might be bypassed by combining the two axioms into a single one. But it would introduce the other flaw – information loss. No new target facts could be generated if only one of the two predicates appears in the source dataset, because there was no matching axiom then could be applied to. One meaning definition axiom is still required for each predicate in the deep ontology.

We can solve the problem by rewriting the axioms with explicit term-generating functions instead of the existential quantifiers. Using the explicit functions can give us finer control over term generation than skolemization would. The following two axioms are for the translation from the deep ontology to the surface ontology. The axioms in the other direction can be got by just replacing the "iff" with "if" in the original ones.

```
(forall (si - @JobPos:SchoolOrInstitution
         re - @JobPos:Region)
  (if (@Resume:schoolState si re)
    (and (@JobPos:hasLocationSummary si (@control:aLocSummary si))
         (@JobPos:hasRegion (@control:aLocSummary si) re)))))

(forall (si - @JobPos:SchoolOrInstitution
         m - @JobPos:Municipality)
  (if (@Resume:schoolCity si m)
    (and (@JobPos:hasLocationSummary si (@control:aLocSummary si))
         (@JobPos:hasMunicipality (@control:aLocSummary si) m))))
```

where *@control:aLocSummary*[1] is defined by

```
(:functions (@control:aLocSummary @JobPos:SchoolOrInstitution)
              - @JobPos:LocationSummary)
```

With these new axioms, only one `(aLocSummary SomeInstitution)` is created for the same `SomeInstitution`. And finally, just one instance of *@JobPos:LocSummary* will take the place in the target dataset.

[1] We use the prefix @control as a convention our inference engine requires for the term-generating functions.

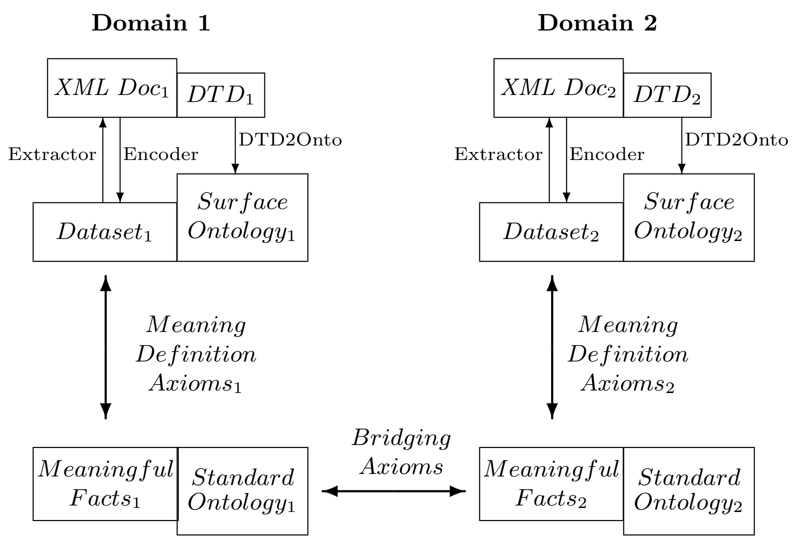

Fig. 3. translating across domains

Besides translating between XML and RDF (DAML+OIL, OWL), our approach also makes it possible to translate between heterogeneous XML specifications. If two XML specifications are describing the same domain, only one deep standard ontology of this domain is needed. We can use one set of meaning definition axioms to represent the information contained in one XML specification in a dataset of the deep ontology, and then use a different set of meaning definition axioms to translate that dataset back to the other XML specification. If the two specifications are about two different but overlapping domains, two deep ontologies are needed. We just carry out one more ontology translation between the two deep ones. The latter case is shown in Fig. 3.

5 Related Work and Discussion

The Semantic Web Project has recently proposed the Web Ontology Language, *OWL*. Derived from RDF and DAML+OIL, OWL provides a rich set of constructs for publishing and sharing ontologies on the World Wide Web. However, users of RDF, DAML+OIL or OWL have to build up their ontologies and datasets based on the RDF specification. XML documents that do not follow the RDF specification are beyond the capability of Semantic Web agents. Our work can serve as a bridge between the semantical RDF and the structural XML. Through the meaning definition axioms, information in XML documents can be accessed by these automated agents, making them more powerful.

As we have pointed out in section 2, XML documents can present meanings of three kinds: objects, their attributes and associations. Past XML transformation tools based on XSLT [4] can capture the meanings of the first two, but most of them neglect the meanings of associations between objects, which should be the more important component in constructing the whole structure of an ontology.

Worden's Meaning Definition Language (MDL) [12] is designed to enable tools to access XML at the level of meaning rather than its structure. An MDL file defines what an XML document means in terms of a UML class model or RDF Schema, and defines how it expresses the meaning by using XPath. MDL gives the mappings between XML vocabularies and the underlying conceptual model, and is able to capture the three kinds of meanings. Generally, objects of classes correspond to XML elements. Attributes of objects are represented by attributes of XML elements. Associations between objects can be represented in various ways within XML vocabularies, and each is handled specifically. In our approach, the meaning definition axioms also capture the three kinds of meanings in XML documents. Unlike an MDL file, our axioms are represented as logic formulas. In our view, meanings of associations between objects can be stated more explicitly using first order logic than using XPath. It is also much easier to build and understand an axiom both of whose sides are in the same logical syntax.

Researchers at the National Taiwan University [18] proposed their logic-based framework of a reasoning system for unifying heterogenous XML documents. The Path Inference Language (PIL) they developed is specially designed for the tree-structure of XML. Logical facts are drawn from XML documents by the semantic extractor which contains a set of extraction rules. An extraction rule is written in PIL and is composed of a matching part and a transformation part. The matching part is used to locate nodes in the XML documents and the transformation part is used to transform the located nodes into logical facts. Further reasoning can be done based on the rules in the common domain ontology, which is also in PIL syntax. Our approach first extracts a set of logical facts automatically to the surface ontology, then rewrites it with the deep standard ontology to make the semantic meanings more explicit. The meaning definition axioms differ from PIL extraction rules in that they can be used in both translation directions. Automated inference can translate data forward and backward between XML languages and the Semantic Web ontologies.

6 Conclusion

In this paper, we extended our ontology translation service, *OntoMerge*, beyond our initial focus on translating ontologies and datasets written in DAML+OIL. The huge web resources available in XML are a tempting target. Since the release of XML in 1998, thousands of XML specifications have been developed by various communities. Massive data are presented in XML format. This huge resource can provide us with plenty of test cases.

In an XML document, the meanings hide in the specific structure of the document or even in the text of those tags. This makes it hard for automated agents to understand XML documents without the intervention of human beings. In this paper, we presented our approach of assigning meanings to XML documents through the Meaning Definition Axioms. We first build up the surface ontology from the DTD file automatically, which captures the default semantics of the XML specification. The default semantics might be plausible, but in most cases it would mix semantic knowledge with syntactic stuff. The deep standard ontology is used to separate these two because it has the advantage of concerning itself entirely with the domain knowledge. The meaning definition axioms are written by domain experts to bridge between the surface and the deep ontologies. They also provide the flexibility for either side to evolve independently. With these axioms, automated reasoning can lift XML data up to the ontology level and serialize ontology datasets to XML format.

We have tested our translation system with two XML specifications in the domain of Resume. It works well so far. Constructing the meaning definition axioms needs the help from domain experts, and it usually takes days to develop them depending on the size of the XML specification and how the semantic relationships will be. We will try more domains to justify the practicality of our approach.

References

1. Ora Lassila and Ralph R. Swick. Resource Description Framework(RDF) Model and Syntax Specification. W3C Recommendation, 22 February 1999. Available at http://www.w3.org/TR/REC-rdf-syntax
2. Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft, 23 January 2003. Available at http://www.w3.org/TR/rdf-schema/
3. Frank van Harmelen, Peter F. Patel-Schneider and Ian Horrocks. Reference description of the DAML+OIL (March 2001) ontology markup language, 2001. Available at http://www.daml.org/2001/03/reference.html
4. James Clark. XSL Transformations (XSLT) Version 1.0, W3C Recommendation, 16 November 1999. Available at http://www.w3.org/TR/xslt
5. Web-Ontology (WebOnt) Working Group. http://www.w3.org/2001/sw/WebOnt/
6. http://www.hr-xml.org/schemas/dtd/recruiting/jobpositionseeker-v1.0.dtd
7. DAML Ontology Library. http://www.daml.org/ontologies/
8. Mike Dean. DAML Country Codes. http://www.daml.org/2001/09/countries/
9. http://orl01.drc.com/daml/Ontology/Resume/3.1/Resume-ont.daml
10. OntoMerge, Ontology Translation by Merging Ontologies. http://www.cs.yale.edu/homes/dvm/daml/ontology-translation.html
11. PDDAML, An Automatic Translator Between PDDL and DAML http://www.cs.yale.edu/homes/dvm/daml/pddl_daml_translator.html
12. Robert Worden. Meaning Definition Language, Version 2.06 Available at http://www.charteris.com/XMLToolkit/Downloads/MDL206.pdf
13. Kim Bartkus. Staff Exchange Protocol(TM) 1.1. Recommendation, HR-XML Consortium.
14. Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, pages 34–43, May 2001.
15. Drew McDermott and Dejing Dou. Reprsenting Disjunction and Quantifiers in RDF. In *Proceedings of International semantic Web Conference 2002*, pages 250–263, 2002

16. Dejing Dou, Drew McDermott, and Peishen Qi. Ontology Translation by Ontology Merging and Automated Reasoning. In *Proc. EKAW02 Workshop on Ontologies for Multi-Agent Systems*, 2002. Available at http://www.cs.yale.edu/homes/dvm/papers/DouMcDermottQi02.ps
17. Drew McDermott. The Planning Domain Definition Language Manual. Technical Report 1165, Yale Computer Science, 1998. (CVC Report 98-003).
18. Yuh-Pyng Shieh, Chung-Chen Chen, and Jieh Hsiang. A Reasoning Framework for Heterogeneous XML. International Conference on Information Technology: Research and Education (ITRE2003), Newark, New Jersey, August 10-13, 2003.
19. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Inc, 1995.