Derivation of OWL Ontology from XML Documents by Formal Semantic Modeling

Shihan Yang1,2, Jinzhao Wu1, Anping He1, Yunbo Rao3
1School of Information Sciences and Engineering, Guangxi University for Nationalities, Nanning, China, 530006
2Chengdu Documentation and Information Center, Chinese Academy of Sciences, China, 610041
3School of information and software engineering, University of Electronic Sci.&Tech. of China, Chengdu, 610054
Email: {dr.yangsh, hapetis, cloudrao}@gmail.com; himrwujz@yahoo.com.cn

Abstract—The extensible markup language (XML), a standard format of web information, has a clear syntax but unfortunately an ambiguous formal semantics, which results in being not used directly in semantic web applications. So it is tough job to reuse XML-based data intelligently in the semantic web. To address this problem, a new formal technique of obtaining ontology data automatically from XML documents is proposed. We provide the XML a semantical interpretation by developing a graph-based formal language, which then can be automatically mapped into web ontology language OWL with semantics preserved. The semantic validity and entailment problem are also concerned. The automatical mapping tool has also been developed.

Index Terms—semantic web, ontology, XML, OWL, formal semantics

I. INTRODUCTION

Extensible markup language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable, but machine-understandable. XML documents are delivered to most transport and store data on the Internet. Most applications of the semantic web, such as semantic-based integrations, intelligent web searching, and internet based knowledge reasoning, can not use XML documents directly, due to that XML formally governs syntax only but not semantics.

There are several ways to address this problem:

- Transforming XML documents into ontology: Techniques of artificial intelligence are always adopted to discover knowledge from XML documents, such as pattern discovery [1], text mining [2], classify [3] [4], and fuzzy computing [5]. Those extracting semantic information are just a little part of meaning in XML documents, and are not enough to provide semantic web applications with ontology data. On the other hand, [6] develops a set of mapping rules between XML schemata and ontology to transforming XML documents into ontology directly. The mapping rules are strongly related to XML and ontology specification, i.e., it should be changed frequently when XML schemata or ontology languages are changed.
- Researching and reasoning semantically on XML documents by annotating XML documents with semantical information:
  A lot of annotating techniques are used to extracting knowledge from XML documents [7] [8]. Before annotating XML documents, [9] computes semantic similarity between them for an accurate purpose. [10] annotates XML with domain ontology in order to build XML knowledge base. [11] decides whether there exists a semantics-preserving mapping between two XML schemata by defining the semantics of XML data by means of a semantic annotation based on the specific ontology. The annotating technique needs to change XML documents themselves or adds more information to them for semantically annotating. Usually, changing them or giving additions to them is impossible for most legacy XML documents.
- Adding formal structure to unstructured or semi-structured data (XML documents) for reasoning purpose:
  The literature [12] develops a middle formal language to describe semi-structured data for model checking purpose. [13] and [14] employ graph-based formalism to model semi-structured data and query on it based on the fixed point computation. [15] proposes a labeled graph schema to represent semi-structured data, and [16] extended that with constraints. This method focuses on semantic operations on XML documents, but not on semantic data implied in them.

Our purpose is to transform legacy XML documents into ontology for most semantic web applications automatically or semi-automatically. In order to handle legacy XML documents without changing them, we focus on the semantic data hidden in XML documents, and then propose a new technique to transform XML documents to web ontology language OWL2 [17] through middle graph-based formal language, which expression is between XML schemata and OWL2. We do not directly build mapping rules between XML and ontology, but transform XML into a middle language model, then
automatically transform the model into ontology. This indirectly transforming decouple dependence between mapping rules and mapping sources/targets. Here the formal language is used as formal semantically interpreting XML documents, but not as reasoning and computing on them. In fact, we employ formal language as middle semantic modeling language, which can be automatically transformed into OWL2, a standard web ontology language.

In the rest of this paper, section II introduces the graph-based formal language W-graph. Section III gives XML documents to OWL ontology is developed in section IV, and section V shows an automatically transforming tool to obtain ontology from XML documents. Section VI concludes and discusses the future works.

II. GRAPH-BASED FORMAL LANGUAGE

W-graph is a simple graph-based formal language that we can use to express instances and schemata of data set. The language here is used as a semantically modeling language for XML documents.

A. Syntax

The following definition distinguishes different two kinds of nodes in the original W-graph [12] definition.

Definition 1: A W-graph $G_w$ is a directed labeled graph $(N,E,L)$, where $N = (N_a, N_c)$ is a finite set of nodes, $N_a$ a finite set of atomic nodes, depicted as ellipses, $N_c$ a finite set of composite nodes, depicted as rectangles, $E \subseteq N \times (\mathcal{G} \times \mathcal{L}) \times N$ is a set of labeled edges of the form $(m, \text{attribute}, n)$, $\ell$ is a function $\ell : N \to \mathcal{G} \times (\mathcal{L} \cup \{\perp\})$, $\mathcal{G}$ is a set of labels, and $\perp$ is a symbol for nothing (empty label), read as bottom.

Nodes in W-graph always represent objects, and edges represent relationships between nodes. There are two types of concrete W-graph: instances and schemata. An instance can be formally defined as the following.

Definition 2: A W-instance $I$ is a W-graph such that $\ell_I(e) = \text{solid}$ for each edge $e$ of $I$ and $\ell_I(n) = \text{solid}$, $\ell_I(n) \neq \perp$ for each node $n$ of $I$.

In Fig.1 a W-instance is depicted. It describes information that two teachers, one 37 years old, one 40 years old, both of them teach database course, and the student Smith attends the same course. In W-graph, edge attributes are made by two components, the color and the label, and the function $\ell$ return a color and a label (possibly empty, $\perp$) for each node. Edge labels are written close to the corresponding edges, and node labels are written inside the rectangles representing the nodes. The set of colors $\mathcal{G}$ denotes how the lines of nodes and edges are drawn (solid or dashed), and we also call this information the color of a node or edge. On the other hand, the function $\ell$ can be seen as the composition of the two single valued functions $\ell_c$ and $\ell_L$, so $\ell$ can be implicitly defined also on edges: if $e = (m, (c, k), n)$, then $\ell(e) = c$ and $\ell_L(e) = k$.

Two nodes $S, T \subseteq N$, $T$ is accessible from $S$ if for each node $n \in T$ there is a node $m \in S$ such that there is a path in W-graph $G_w$ from $m$ to $n$. For example, in the W-instance $I$ of Fig. 1, the set $\{n_1, n_5, n_6, n_7, n_8\}$ is accessible from the set $\{n_1, n_2, n_3\}$.

B. Bisimulation-based Semantics

In this subsection, we observe a bisimulation-based semantics for W-graph. The bisimulation provides us the semantic-preserving mapping between XML documents and ontology as long as they are precisely encoded by W-graph language.

Definition 4: Given two W-graphs $G_0 = \langle N_0, E_0, \ell_0 \rangle$ and $G_1 = \langle N_1, E_1, \ell_1 \rangle$, a relation $b \subseteq N_0 \times N_1$ is said to be a bisimulation between $G_0$ and $G_1$ if and only if:

1) for $i = 0, 1$, $\forall n_i \in N_i$, $\exists n_{i-1} \in N_{i-1}$ such that $n_0b_m n_1$

2) for $\forall n_0 \in N_0$, $\forall n_1 \in N_i$, $\exists n_{i-1}$ such that $n_0b_n n_1 \sim \ell_0^L(n_0) \wedge \ell_1^L(n_1) = \perp$ or $\ell_0^L(n_0) = \perp \wedge \ell_1^L(n_1)$, and

3) for $i = 0, 1$, $\forall n_i \in N_i$, let

$M_i(n_i) = \{ (m, \text{label}) : (n_i, (\text{color}, \text{label}), m) \in E_i \}$.

Then, $\forall n_0 \in N_0$, $\forall n_1 \in N_1$ such that $n_0b_m n_1$, for $i = 0, 1$, it holds that $V(m_i, \ell_i) \in M_i(n_i)$, $\exists (m_{i-1}, \ell_{i-1}) \in M_{i-1}(n_{i-1})$ such that $m_0b_m n_1 \wedge \ell_i = \ell_{i-1}$.

Write $G_0 \bowtie G_1$ ($G_0 \bowtie\bowtie G_1$) if $b$ is (not) a bisimulation between $G_0$ and $G_1$. Write $G_0 \sim G_1$ ($G_0 \bowtie\bowtie G_1$) if there is
(not) a bisimulation between $G_0$ and $G_1$, in this case also say that $G_0$ is bisimilar to $G_1$.

Condition (1) says that no node in the two graphs can be left out of the relation $b$. Condition (2) says that two nodes belonging to relation $b$ have exactly the same label, else than the case of dummy nodes, labeled by $\bot$. Condition (3) deals with edge correspondence. If two nodes $n_0, n_1$ are in relation $b$, then every edge having $n_0$ as endpoint should find as a counterpart a corresponding edge with $n_1$ as endpoint. Notice that output values of the $e_G$ function (solid / dashed) are not taken into account in the bisimulation definition.

Based on the bisimulation semantics, we can describe how a $W$-instance is an instance of a $W$-schema as follows.

Definition 5: A $W$-instance $I$ is an instance of a $W$-schema $S$ if $\exists I' \ni I$, s.t., $S \sim I'$. $S$ is also said to be a schema for $I$. $I'$ is said to be a witness of the relation schema-instance.

Figure 2 is an example. $S$ is a schema for $I$ (an instance over schema $S$). To build the witness $I'$, add to $I$ an edge labeled by works linking the entity node Person of Bob with the entity node Town. Moreover, add edges labeled by lives from the two nodes labeled Person to the node labeled Town, and add also an edge reverse to the father edge. It is easy to check that a bisimulation from $S$ to $I'$ is uniquely determined.

C. More Detail Specification

For some concrete transforming tasks, we also give $W$-Graph more specification in order to depict some details. The structure of nodes and edges in $W$-graph is shown in figure 3, where the fields are:

- **Label**: A label of node or edge is also a structure, where Name is a variable-length character string (describing a concept for a composite node, a data type for an atomic node in the schema, a data value for an atomic node in the instance, and a relationship for an edge), Color solid or dashed (the meaning is same as above definition), Annotation a comment on node or edge (be null or a variable-length character string). The label gives some readable information on object (node) or relationship (edge).

- **Type**: The data type of the object’s value. Each type is either an atomic type (such as integer, string, real number, etc.) for an atomic node, a set type for a composite node in the schema (so this mechanism can handle nesting structure), or a concept type (not set type) for a composite node in the instance. The possible concept type must be already occurred in the corresponding schema described by Name field of Label field of one composite node, and the possible atomic types are not fixed and may vary from information source to information source (here XML documents).

- **Value**: A variable-length value for the object. For composite node, a value is either a set of composite nodes in the schema, or a individual of the concept type in the instance. For atomic node, it is null in the schema, but not null in the instance.

- **Flag**: A flag is used to distinguish different the type of node, 1 for composite node, 0 for atomic node.

- **OID**: A unique variable-length identifier for the object.

- **I_OID**: The OID of node, which edge is from.

- **O_OID**: The OID of node, which edge is to.

- **RID**: A unique variable-length identifier for the relationship.

We use the conventional dot separate expression to refer to the field of nodes or edges. For example, $N.Label.Name$ expresses the name of the label for a node $N$, $M.Value.OID.Label$ says the label of a node whose OID is in the set / list of value field of the node $M$.

III. XML Semantically Interpreting

XML documents record implicitly rich semantic information, but the information cannot be understood by machines due to absence of formal semantic definition. In this paper, the W-graph is used to interpret formally XML documents, and the interpretation is explicit. The $W$-
schema interprets the XML schemata, and the W-instance interprets concrete XML documents.

Definition 6: (Data type) A data type is a tuple \( d = (L, V, \tau) \), where
- \( L \) is a finite set of literal, \( L \neq \emptyset \);
- \( V \) is a set of values;
- \( \tau(d) : L \rightarrow V \) maps a literal with data type \( d \) into a corresponding value.

Furthermore, a map \( \eta(d) : D \rightarrow N_d \) gives each data type \( d \in D \) a name \( n \in N_d \), where \( D \) is a set of data types, and \( N_d \) a set of names of data types.

Definition 7: (XML model) A XML model \( M_{XML} \) is a two-tuple \( M_{XML} = (N, P) \), where
- \( N \) is a finite set of nodes. \( N = N_e \cup N_a \cup N_c \). \( N_e \) is a set of element nodes occurred in XML documents, \( N_a \) a set of attribute nodes, and \( N_c \) a set of data type names;
- \( P \) is a set of binary relationships between nodes, which includes binary relations decided by the order that node occurs in XML documents, \( P \subseteq N \times N \).

So a W-graph interpretation can impose on XML schemata and XML concrete documents. The interpretation can obtain as the following two definitions.

Definition 8: (W-schema interpretation for XML schema) For a XML model \( M_{XML} = (N, P) \), a W-schema \( \mathcal{W} = (N_w, E, \ell) \) is an interpretation based on the following interpretation rule \( I \):
- \( N_w = N_e \), where \( N_w = N_{\ell} \), i.e., data type node in XML should be interpreted as atomic nodes in W-schema;
- \( E = P \);
- \( \ell \) function satisfies following conditions:
  - \( \ell_{\ell}(e) = \text{solid}, \ell_{\ell}(n) = \text{solid}, \ell_{\ell}(n) = \perp \);
  - \( \ell_{\ell}(e) = \text{hasvalue}', \text{if } p \in P, p \in N_{\ell} \times N_{\ell} \);
  - \( \ell_{\ell}(e) = \text{hasattribute}', \text{if } p \in P, p \in N_{\ell} \times N_{\ell} \);
  - \( \ell_{\ell}(e) = \text{relate}', \text{others} \).

Definition 9: (W-instance interpretation for XML document) For a XML model \( M_{XML} = (N, P) \), a W-instance \( \mathcal{W} = (N_w, E, \ell) \) is an interpretation based on the following interpretation rule \( I' \):
- \( N_w = N_e \), where \( N_w = N_{\ell} \), i.e., data type node in XML should be interpreted as atomic nodes in W-instance, and \( N_w \cup N_a \cup N_c \), i.e., element nodes and attribute nodes in XML should be interpreted as composite nodes in W-instance;
- \( E = P \);
- \( \ell \) function satisfies following conditions:
  - \( \ell_{\ell}(e) = \text{solid}, \ell_{\ell}(n) = \text{solid}, \ell_{\ell}(n) = \tau(d), d \in N_{\ell} \), this is only different from above definition, due to each value of elements should be assigned in the W-instance;
  - \( \ell_{\ell}(e) = \text{hasvalue}', \text{if } p \in P, p \in N_{\ell} \times N_{\ell} \);
  - \( \ell_{\ell}(e) = \text{hasattribute}', \text{if } p \in P, p \in N_{\ell} \times N_{\ell} \);
  - \( \ell_{\ell}(e) = \text{relate}', \text{others} \).

So the XML documents could be semantically interpreted with W-graph in terms of the above two definitions. We must notice that the semantical interpretation just transforms the XML model (a tree model) into a W-graph model (a simple graph model) automatically, which is not enough, due to most semantic information still not be drawn out. For solving this problem, we present a visual editor for W-graph language (should be introduced in section V) in order to adjust these semantic information by GUI (Graphic User Interface), so a formally operations on W-graph should be defined here.

Definition 10: Following operators could be imposed on the W-schema to change it.
- **Abstracting** : For three composite nodes \( m, n, d \in N_c \), if \( (m, d), (n, d) \in E \), and \( \forall v \in N_c - \{m, n\}, (v, d), (d, v) \notin E \), then drop the node \( d \) and atomic nodes associated with it (drop the edges associated with these nodes too), and add two edges \( (m, n), (n, m) \) to the W-schema and labeled them uniquely. In practice, some nodes and edges are deleted from the W-schema, and two new edges are added into it. We call this operator Abstracting.
- **Supplement** : For a composite node \( n \in N_c \), some new composite nodes \( n_1, \ldots, n_i \) can be added for supplementing it, and these nodes can only have their own atomic nodes, and \( (n, n_1), \ldots, (n, n_i) \) are added to \( E \), labeled these edges supplement.
- **Rename** : The name of any element in W-schema can be renamed in order to make the element more meaningful.

The operators do not reduce the original semantics of the schema. Intuitively, the abstract can delete some nodes without reducing basic information, the supplement can add some new nodes limitedly, and the rename can make the meaning of elements more precise.

IV. MAPPING INTO ONTOLOGY

In this section, a mapping function \( \pi \) from W-graph to OWL ontology is provided after introducing simply OWL abstract syntax and semantics, which a transforming algorithm can be based.

A. OWL DL Ontology

OWL DL, the description logic style of using OWL, is based on description logic \( SHOIN(D) \).\(^{18, 19} \) OWL DL can form descriptions of classes, data types, individuals, data values, and axioms that provide information about them.

Definition 11: A OWL DL Ontology is a two-tuple \( O = (ID, A) \), where
- \( ID = ID_c \cup ID_{OP} \cup ID_I \cup ID_{DP} \cup ID_{DR} \) is a finite set of OWL DL identifiers, and \( ID_c, ID_{OP}, ID_I, ID_{DP}, ID_{DR} \) are pairwise disjoint.
- \( ID_c \) is a set of class identifiers, \( ID_{OP} \) a set of object property identifiers, \( ID_I \) a set of individual identifiers, \( ID_{DP} \) is a set data type identifiers, and \( ID_{DR} \) a set of data range identifiers. Each identifier is a URI (Uniform Resource Identifier) reference, which consists of an absolute URL (Uni-
form Resource Location) or prefix, so-called namespace, and a fragment identifier. For example, the qualified name owl:Thing for the URI reference http://www.w3.org/2002/07/owl#Thing.

- \( A = A_C \cup A_P \cup A_I \) is a finite set of OWL DL axioms, and \( A_C, A_P, A_I \) are pairwise disjoint. \( A_C \) is a set of class axioms, \( A_P \) a set of property axioms, and \( A_I \) a set of individuals (also called calls).

In table I, the first column gives the OWL DL abstract syntax, while the second column gives the standard description logic syntax.

**Definition 12:** For an OWL DL ontology \( O = (ID, A) \), an interpretation \([19]\) of it is \( I = (\Delta^I, \Delta^I_D, \cdot) \), where \( \Delta^I \cap \Delta^I_D = \emptyset \), and

- \( \Delta^I \) is the set of individuals, called individual domain of the interpretation.
- \( \Delta^I_D \) is the set of data values, called data-value domain.
- \( \cdot \) is an interpretation function, which is defined in the table I. The interpretation function maps classes into subsets of \( \Delta^I \), individuals into elements of \( \Delta^I \), data types into subsets of \( \Delta^I_D \), data values into elements of \( \Delta^I_D \), object properties into subsets of \( \Delta^I \times \Delta^I \) and the data type properties into subsets of \( \Delta^I \times \Delta^I_D \).

In table I, the third column gives the interpretation-based semantics of OWL DL language, where a symbol \( \# \) denotes the cardinality of a set.

**B. Mapping W-graph into OWL**

The W-graph can be automatically transformed into an OWL2 ontology. We build a formal mapping

\[
\pi : \Sigma_{W\text{-graph}} \rightarrow \Sigma_{OWL}
\]

as follows, \( \Sigma_{W\text{-graph}}, \Sigma_{OWL} \) be finite sets of alphabet of the graph and of the ontology respectively.

**Definition 13:** For a given W-graph \( I = (N, E, L) \) with specification defined before, it can be formally mapped into a OWL \( O = (ID, A) \) according to the following function \( \pi \):

- \( \pi(n_c, Label.Name) = id_c, n_c \in N_c \subseteq N, id_C \in ID_C \subseteq ID, \) when \( n_c.Flag = 1, n_c.Type = set; \)
- \( \pi(n_a, Label.Name) = id_{DR}, n_a \in N_a \subseteq N, id_{DR} \in ID_{DR} \subseteq ID, \) when \( n_a.Flag = 0, n_a.Value = null; \)
- \( \pi(e_c, Label.Name) = id_{OP}, e_c \in N_c \times N_c, id_{OP} \in ID_{OP} \subseteq ID; \)
- \( \pi(e_a, Label.Name) = id_{OP}, e_a \in N_c \times N_a, id_{OP} \in ID_{OP} \subseteq ID; \)
- \( \pi(n_c, Value) = id_c, n_c \in N_c \subseteq N_c \subseteq ID, \) when \( n_c.Flag = 1, n_c.Type \neq set; \)
- \( \pi(n_a, Value) = id_{OP}, n_a \in N_a \subseteq N_a \subseteq ID, \) when \( n_a.Flag = 0, n_a.Value = null; \)
- \( \pi(n_a, Value) = literal in n_a^L \subseteq ID_{OP}, \) when \( n_a.Flag = 0, n_a.Value = null; \)
- \( \pi(e_a) = a_{OP} \subseteq A \subseteq N_c \times N_a, \) and the axiom like this:

\[
\text{DatatypeProperty}(a_{OP}
\begin{align*}
\text{domain}(\pi(e_a, L_OID.Label.Name)) \\
\text{range}(\pi(e_a, O_OID.Type))
\end{align*}
\]

[Functional], where Functional is optional:

- \( \pi(e_{cc}) = a_{OP} \in A_P, e_{cc} \in N_c \times N_c \), and the axiom like this:

\[
\text{ObjectProperty}(a_{OP}
\begin{align*}
\text{domain}(\pi(e_{cc}, L_OID.Label.Name)) \\
\text{range}(\pi(e_{cc}, O_OID.Label.Name))
\end{align*}
\]

[Functional/InverseFunctional], where Functional/InverseFunctional is optional:

- \( \pi(|e_{cc} | for an_c \in N_c, 1 \leq i \leq l, \forall n_a \in N_a, e_{cc} = (n_c, n_a)) \) is \( a_C \in A_c \subseteq A, e_{cc} \in N_c \times N_a \), and the axiom like this:

\[
\text{Class}(a_c \text{ partial restriction}(\pi(n_n, Label.Name))
\begin{align*}
\text{allValuesFrom}(\pi(n_n, Type))
\end{align*}
\]

... restriction(\( \pi(n_n, Label.Name) \) allValuesFrom(\( \pi(n_n, Type()) \));

- \( \pi(|e_{cc} | for an_n \in N_n, and \_Type = set, 1 \leq i \leq k, \forall n_c \in N_c, e_{cc} = (n_c, n_n)) \) is \( a_C \in A_c \subseteq A, e_{cc} \in N_c \times N_n, \) and axioms like these:

\[
\text{subClassOf}(\pi(n_n, Label.Name) \pi(n_c, Label.Name))
\]

... subClassOf(\( \pi(n_n, Label.Name) \) \( \pi(n_c, Label.Name) \)) or like these:

\[
\text{Class}(\pi(n_n, Label.Name) \text{ partial}
\begin{align*}
\pi(n_c, Label.Name)
\end{align*}
\]

... Class(\( \pi(n_n, Label.Name) \) partial \( \pi(n_c, Label.Name) \));

- \( \pi(n_c) = a_i \in A_i, n_c \in N_c \), when \( n_c,Flag = 1, n_c.Type \neq set, \) and the axiom like this:

\[
\text{Individual}(a_i \text{ type}(\pi(n_n, Label.Name))
\begin{align*}
\text{value}(\pi(n_n, Value));
\end{align*}
\]

- \( \pi(n_a) = a_i \in A_i, n_a \in N_a \), when \( n_a,Flag = 0, n_a,Value \neq null \) and the axiom like this:

\[
\text{Individual}(a_i \text{ type}(\pi(n_n, Label.Type))
\begin{align*}
\text{value}(\pi(n_n, Value)).
\end{align*}
\]

Intuitively the above definition maps each node in the W-schema into a class description of the ontology, each type of atomic nodes into a data type, each composite node of instance into a individual assertion, each atomic node of instance into a property assertion, each value of atomic node into literal (value of data type), each edge between composite nodes into a corresponding class axiom, each edge between atomic node and atomic node into a corresponding property axiom, and each annotation into annotation property of the ontology.

**Theorem 1:** The mapping from W-graph language to OWL ontology by means of W-graph does not reduce the semantics.

**Proof:** There are two facets to be proved. Firstly, after XML documents are interpreted semantically into W-schemata and W-instances according to definition 8, 9, the W-graph could be dynamically changed only by operators defined in the definition 10, so we prove that these operators do not changed the semantics according
<table>
<thead>
<tr>
<th><strong>Abstract Syntax</strong></th>
<th><strong>DL Syntax</strong></th>
<th><strong>Semantics</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description (C)</strong></td>
<td>A (URI reference)</td>
<td>A' \in \Delta'</td>
</tr>
<tr>
<td>owl:Thing</td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>owl:Nothing</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>intersectionOf(C₁, C₂)</td>
<td>C₁ \cap C₂</td>
<td>C₁' \cap C₂'</td>
</tr>
<tr>
<td>unionOf(C₁, C₂)</td>
<td>C₁ \cup C₂</td>
<td>C₁' \cup C₂'</td>
</tr>
<tr>
<td>complementOf(C)</td>
<td>¬C</td>
<td>(¬C') = \Delta'</td>
</tr>
<tr>
<td>oneOf(o₁, ⋯, oₙ)</td>
<td>{o₁, ⋯, oₙ}</td>
<td>{o'₁, ⋯, o'ₙ}</td>
</tr>
<tr>
<td>restriction(R someValueFrom(C))</td>
<td>\exists R.C</td>
<td>(\exists R.C)' = [x \mid \exists y. (x, y) \in R '\land y \in C']</td>
</tr>
<tr>
<td>restriction(R allValueFrom(C))</td>
<td>\forall R.C</td>
<td>(\forall R.C)' = [x \mid \forall y. (x, y) \in R ']</td>
</tr>
<tr>
<td>restriction(R minCardinality(n))</td>
<td>\geq nR</td>
<td>(\geq nR)' = [x \mid \exists y. (x, y) \in R ']</td>
</tr>
<tr>
<td>restriction(R maxCardinality(n))</td>
<td>\leq nR</td>
<td>(\leq nR)' = [x \mid \exists y. (x, y) \in R ']</td>
</tr>
<tr>
<td>restriction(U someValueFrom(D))</td>
<td>3U.D</td>
<td>(3U.D)' = [x \mid \exists y. (x, y) \in U' \land y \in D']</td>
</tr>
<tr>
<td>restriction(U allValueFrom(D))</td>
<td>4U.D</td>
<td>(4U.D)' = [x \mid \forall y. (x, y) \in U' \land y \in D']</td>
</tr>
<tr>
<td>restriction(U hasValue(v))</td>
<td>U \vdash v</td>
<td>(U \vdash v)' = [x \mid (x, v') \in U']</td>
</tr>
<tr>
<td>restriction(D maxCardinality(n))</td>
<td>\leq nU</td>
<td>(\leq nU)' = [x \mid (x, y) \in U ']</td>
</tr>
</tbody>
</table>

**Data Ranges (D)**

- D (URI reference) | D | D' \subseteq \Delta'_D |
- oneOf(v₁, ⋯, vₙ) | \{v₁, ⋯, vₙ\} | \{v'₁, ⋯, v'ₙ\} |

**Object Properties (R)**

- R (URI reference) | R | R' \subseteq \Delta' \times \Delta' |
- R⁻ | (R⁻)' = (R')⁻ |

**Datatype Properties (U)**

| U (URI reference) | U | U' \subseteq \Delta' \times \Delta'_D |

**Individuals (o)**

- o (URI reference) | o | o' \in \Delta' |

**Data values (v)**

- v (RDF literal) | v | v' = v' \in \Delta' |

**Class Axioms**

| Class(A partial C₁ ⋯ Cₙ) | A \subseteq C₁ \cap ⋯ \cap Cₙ | A' \subseteq C₁' \cap ⋯ \cap Cₙ' |
| Class(A complete C₁ ⋯ Cₙ) | A = C₁ \cap ⋯ \cap Cₙ | A' = C₁' \cap ⋯ \cap Cₙ' |
| EnumeratedClass(A o₁ ⋯ oₙ) | A = \{o₁, ⋯, oₙ\} | A' = \{o'₁, ⋯, o'ₙ\} |
| SubClassOf(C₁ C₂) | C₁ \subseteq C₂ | C₁' \subseteq C₂' |
| EquivalentClasses(C₁ ⋯ Cₙ) | C₁ = ⋯ = Cₙ | C₁' = ⋯ = Cₙ' |
| DisjointClasses(C₁ ⋯ Cₙ) | C₁ \cap C₂ = \emptyset, i ≠ j | C₁' \cap C₂' = \emptyset, i ≠ j |
| Datatype(D) | D' \subseteq \Delta'_D |

**Property Axioms**

| DatatypeProperty(U) | super(U₁) ⋯ super(Uₙ) | U₁ \subseteq Uₙ, Uᵢ \subseteq Uᵢ' |
|                     | domain(C₁) ⋯ domain(Cₙ) | U₁ \subseteq Cᵢ, Uᵢ' \subseteq Cᵢ' |
|                     | range(D₁) ⋯ range(Dₙ) | U₁ \subseteq Dᵢ, Uᵢ' \subseteq Dᵢ' |
|                     | [Functional] | U₁ \subseteq 1U, Uᵢ is functional |
|                     | SubPropertyOf(U₁ U₂) | U₁ \subseteq U₂, Uᵢ \subseteq Uᵢ' |
|                     | EquivalentProperties(U₁ ⋯ Uₙ) | U₁ = ⋯ = Uₙ, Uᵢ = ⋯ = Uᵢ' |
| ObjectProperty(R) | super(R₁) ⋯ super(Rₙ) | R₁ \subseteq Rₙ, Rᵢ \subseteq Rᵢ' |
|                    | domain(C₁) ⋯ domain(Cₙ) | R₁ \subseteq Cᵢ, Rᵢ \subseteq Cᵢ' |
|                    | range(C₁) ⋯ range(Cₙ) | R₁ \subseteq Dᵢ, Rᵢ \subseteq Dᵢ' |
|                    | [inverseOf(Rᵢ)] | Rᵢ = (\neg Rᵢ)', (Rᵢ')⁻ |
|                    | [Symmetric] | R₁ = Rₙ, Rᵢ = (Rᵢ')⁺ |
|                    | [Functional] | U₁ \subseteq 1R, Rᵢ \subseteq (Rᵢ')⁻ is functional |
|                    | [InverseFunctional] | Tr(R) \subseteq (Rᵢ')⁺ is functional |
|                    | [Transitive] | R₁ \subseteq Rₙ, Rᵢ \subseteq Rᵢ' |
|                    | SubPropertyOf(R₁ R₂) | R₁ \subseteq R₂, Rᵢ \subseteq Rᵢ' |
|                    | EquivalentProperties(R₁ ⋯ Rₙ) | R₁ = ⋯ = Rₙ, Rᵢ = ⋯ = Rᵢ' |
| AnnotationProperty(S) | | |

**Facts**

| Individual(o type C₁ ⋯ type Cₙ) | o \in Cᵢ, o' \in Cᵢ' |
| value(Rᵢ o₁ ⋯ value(Rᵢ oₙ) | (o₁, oₙ) \in Rᵢ, (o₁', oₙ') \in Rᵢ' |
| value(U₁ v₁ ⋯ value(U₁ vₙ) | (o₁, oₙ) \in U₁, (o₁', oₙ') \in U₁' |
| SameIndividual(o₁ ⋯ oₙ) | o₁ = ⋯ = oₙ, o'₁ = ⋯ = o'ₙ |
| DifferentIndividuals(o₁ ⋯ oₙ) | oᵢ ≠ oⱼ, oᵢ' ≠ oⱼ' |

© 2013 ACADEMY PUBLISHER
to the bi-simulation semantics defined in the definition 4. For Abstract operator, two 1:1 relationships are reduced to a m:n relationship, and the information of that is recorded in the new two edges; for Supplement operator, more semantic information are added, and nothing is reduced; and for Rename, no formal semantic information is changed except for element renamed for more readable.

Secondly, for a legal W-graph $G_W$ and a mapped OWL ontology $O = \pi(G_W)$ (defined in the definition 13), we prove that a mapping $\alpha : G_W \rightarrow G_O$ must be existed, such that $G_O = \alpha(G_W)$ is a model of $O$ ($G_O$ is an interpret of the ontology $O$). We just let $\alpha = G \circ \pi$, where $\circ$ is the compound operator of mapping, $G$ an interpret function of $O$.

The theorem shows that it does not reduce the semantics when transforming from the semantic interpretation of XML documents to OWL ontology through the medium formal language W-graph.

V. Automatically Transforming Tool

Based on definition 13, a tool has been developed to transform XML documents into ontology automatically. The processing of the tool is shown in figure 4. XML documents should be interpreted semantically into W-graph files by the Interpreter, and then these W-graph files could be edited in the Editor with GUI. During the process of editing, the Checker should execute syntax checking and partial semantic checking for W-graph files. At last, the Convertor transforms those well-defined W-graph files into OWL2 ontology files automatically.

The Interpreter is based on definition 6, 7, 8, and 9. It discovers some semantics from the structure and the contents of XML documents, which semantics information are objective, and maybe not fit to some semantics applications. Then the Editor leaves a choice to revise these objective semantics information. The Editor is implemented based on operations over W-graph according to definition 10. The semantics information is subjective after editing, and the Checker is used to guarantee syntax and partial semantic correctness of these editing. Lastly, the subjective W-graph files are transformed automatically into OWL2 ontology files by the Convertor, which is implemented according to definition 13.

The OWL2 files obtained from XML documents can be validated by the online OWL2 Validator\(^1\). The cost of transforming depends on the size of elements of XML documents, which is finite.

VI. Conclusion and Future Works

Based on graph-based formal language W-graph, the method of transforming XML documents into OWL2 automatically has been proposed. The tool for the transforming has also developed. The formal semantics interpretation has been imposed on XML documents by middle formal language W-graph, then the GUI W-graph editor has been provided to revise semantics information, lastly the W-graph after revised has been automatically transformed into OWL2 by the convertor.

In the future, an existed ontology should be introduced to help W-graph edit meaningfully and automatically in a sense. And results of analyzing by AI methods should be added into the process of interpreting XML documents with formal language W-graph, so more meaningful information should be added to middle models.

Acknowledgment

This research is partially supported by the NNSF of China under Grant No. 60873118 and 60973147, the NSF of Guangxi under Grant No. 2011GXNSFA18154, the Sci.&Tech. Foundation of Guangxi under Grant No. 10169-1, Guangxi Scientific Research Project No.201012MS274. This research is also partially supported by the West Light Foundation of the CAS on the project: Automatically extracting semantic metadata of digital document resources on science and technology, by the starting fund of GXUN under Grant No.2011QD017, by the grant of Guangxi Key Laboratory of HCIC Open Fund (HCIC201110), and the Fundamental Research Funds for the Central Universities of Lanzhou University (No.860772).

References


\[5\] http://owl.cs.manchester.ac.uk/validator/


