Concept Lattice-Based Semantic Web Service Ontology Merging

Hong Xia
Department of Computer Science and Technology, Xi’an University of Posts and Telecommunications, Xi’an 710121, china xiahong@xupt.edu.cn

Abstract - Using ontology semantically express the service of capabilities, correctly match, discovery and composition service. Domain ontology and Formal Concept Analysis aim at modeling concept. In ontology engineering the FCA role is reusing independently developed domain ontology. The method is intended to support the ontology engineer in difficult activities such as ontology merging in the development of the Semantic Web.

Index Terms - Semantic Web Service, concept lattice, Ontology merging

1. Introduction

Semantic Web extends the current Web in information, aims to add machine-interpreted information to Web in order to provide intelligent access to heterogeneous and distributed information. Ontology concepts are extracted the relative concepts in specify domain and used to define semantic Web service. If we could develop ontology which could be used for multiple systems, then shared and reused a common terminology. So support merging ontology which sharing would be possible even between Web services based on different ontology.

In order to solve the problems we involved in our previous work [1], identify possible relations between pairs of Web services by checking semantic similarities. Using an ontology, we can discover relations between two services even the conditions don’t match each other syntactically.

Both domain ontology and concept lattice aim at modeling concepts. In this paper, provides a method supports the ontology engineer in reusing existing ontology relying on concept lattice. From a theoretical point of view, ontology concepts are identified with FCA concepts [2]. The main contribution of our method is the possibility of using concept lattice theory to deal with the generating ontology. The rest of the paper is organized as follows. Section 2 introduces related useful results; section 3 shows basic concept including concept lattices and domain ontology, section 4 giving ontology merging algorithm and example; finally section 5 provides the conclusions.

II. Related Work

Many researchers consider ontology is a theory of content referring to object types, properties and possible relationships in a specified knowledge domain [3]. In document [4], ontology is the explicit formal specification of items in a domain and the relationships among them. If ontology could be used to the elementary for multiple Web services’ semantic information, they would share common terms and facilitate to share and reuse. We provide a method to support merging ontology which sharing would be possible.

In reference [5,6], the importance of dealing with semantically heterogeneous data by using ontology has been emphasized. In reference [6], the methods and tools supporting ontology integration and maintenance can be divided based on Galois Lattices and Description Logics (DL). Reasoning is generally used in order to compute relations among different information sources [7]. In document [8], an ontology merging method based on similarity relations among concepts represented according to DL. The existing work concerning the combination of domain ontology and concept lattice techniques, one proposal concerning a similarity measure for Concept Lattices will be recalled [9,10].

Formal concept analysis [11,12] is proposed by Wille R in 1982. Concept lattice that is an effective approach to analyze data so has rapidly developed. As formalized tool of data analyzing and knowledge processing, concept lattice theory has been successfully applied to various fields [13,14].

Ontology merging that consists in taking two or more source ontology and returning a merged ontology based on the given source has been investigated in reference [14, 15]. Given two or more source ontology, one context is constructed for each of them, by applying natural language processing techniques.

III. Basic concepts

For given service information table \( K = (S,A,R) \), is called a formal context in formal concept analysis. As table 1 shows, \( S \) is a set of services, \( A \) is a set of attributes, \( R \) is a binary relation between \( S \) and \( A \), \( R \in S \times A \). For service \( s \in S \), attribute \( a \in A \), so \( sRa \) express service \( s \) has attribute \( a \).

In formal context, for set of service \( S \) and set of attribute \( A \) may define the following two function \( f \) and \( g \).

<table>
<thead>
<tr>
<th>S/A</th>
<th>TA₁</th>
<th>TA₂</th>
<th>TA₃</th>
<th>TA₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS₁</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>TS₂</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TS₃</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TS₄</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TS₅</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE 1 example of formal concept
\[ \forall SC, PC \subseteq S : f(SC) = \{ a \in A \mid \forall s \in SC, sRa \} \]
\[ \forall PC \subseteq A : g(PC) = \{ s \in S \mid \forall a \in PC, sRa \} \]

SC expresses service class, PC expresses attribute class, sRa expresses existed relation between \( s \in S \) and \( a \in A \).

From formal context gained each two - tuples \( (SC, PC) \) satisfied \( SC = g(PC) \) and \( PC = f(SC) \) is defined as formal concept.

Concept lattice \( L(K) \) corresponded the formal context \( K = (S, A, R) \) is shown in Figure 1.

![Fig. 1 formal context corresponded the concept lattice](image)

Because concept lattice is represent formal of relationship of between concepts in formal context, it and correspond formal context are one to one correspondence. Therefore, concept lattice distributed processing should involve in divide or merge and so on process about formal context. According to Wille R’s idea [12] and reference [16], we have several definitions as follows.

**Definition 1**

\[ K = (S, A, R), K_i = (S_1, A_1, R_1), \ldots, K_n = (S_n, A_n, R_n) \]

are called formal context.

(1) if \( A_1 = A_2 = \ldots = A_n = A \), then \( K_1, K_2, \ldots, K_n \)’s horizontal union is denoted as:

\[ K_1 + K_2 + \ldots + K_i + \ldots + K_n = (S_1 \cup S_2 \cup \ldots \cup S_i \cup \ldots \cup S_n, A, R_1 \cup R_2 \cup \ldots \cup R_i \cup \ldots \cup R_n) \]

(2) if \( S_1 = S_2 = \ldots = S_i = \ldots = S_n = S \), then \( K_1, K_2, \ldots, K_i, \ldots, K_n \)’s vertical union is denoted as:

\[ K_1 \uplus K_2 \uplus \ldots \uplus K_i \uplus \ldots \uplus K_n = (S, A_1 \cup A_2 \cup \ldots \cup A_i \cup \ldots \cup A_n, R_1 \cup R_2 \cup \ldots \cup R_i \cup \ldots \cup R_n) \]

This definition is that multiple formal context’s horizontal union and portrait union.

**Definition 2** \( K_i = (S_i, A_i, R_i) \) and \( K_j = (S_j, A_j, R_j) \) are two different formal contexts in same service domain, \( K_i \pm K_j = (S_i \cap S_j, A_i \cup A_j, R_i \cup R_j) \) called two formal contexts’ portrait add operation (±).

**Definition 3** In \( K = (S, A, R) \), formal concept \( C_i = (SC_i, PC_i) \) and \( C_j = (SC_j, PC_j) \) (\( i \neq j \))

(1) if \( SC_i = SC_j \) then called \( C_i = C_j \), that is concept equal;

(2) if \( SC_i \supseteq SC_j \), then called \( C_i \) more than \( C_j \);

(3) if \( SC_k = SC_i \cap SC_j \), \( PC_k = PC_i \cup PC_j \), then \( C_k = (SC_k, PC_k) = C_i \pm C_j \), that is portrait add operation between concepts.

**Definition 4** if \( L(K_i) \) and \( L(K_j) \) are two intension consistent concept lattice, suppose their portrait union operation \( L(K_i) \cup L(K_j) \) equal concept lattice \( L(K) \).

**Theorem 1** if \( L(K_i) \) and \( L(K_j) \) are intension consistent concept lattice, then \( L(K_i) \cup L(K_j) = L(K_i \pm K_j) \).

Theorem 1 proof sees reference [16].

Concept lattice portrait union algorithm utilize attribute-based incremental formation algorithm. Add concept \( C = (SC, PC) \) in concept lattice \( L(K) \), first in lattice according to the relation of all node and new increase concept, find required modify concept, when relationship between concepts change, corresponding edge will modify.

For concept \( C \), its intension and extension respectively are denoted \( Intension(C) \) and \( Extension(C) \).

**Definition 5** for concept \( C = (SC, PC) \), in concept lattice \( L(K) \) existed one concept \( C_1 = (SC_1, PC_1) \) satisfied \( SC_1 \subseteq Extension(C) \). Concept \( C_1 \) is update concept of \( C \). After update it is \( (SC_1, Intension(C)) \cup PC_1 \).

**Definition 6** for concept \( C = (SC, PC) \), if concept lattice \( L(K) \) existed one concept \( C_1 = (SC_1, PC_1) \), new increase concept \( C_{new} = (SC_{new}, PC_{new}) \) and satisfied:

(1) \( SC_{new} = Extension(C) \cap SC_1 \) and in lattice does not exist arbitrary concept \( C_2 \) which comes \( Extension(C_2) = SC_{new} \) into existence.

(2) arbitrary sub concept \( C_3 \) of concept \( C_1 \) does not exist \( Extension(C_3) \cap Extension(C) = SC_{new} \) that called concept \( C_1 \) and concept \( C \) are generation sub concept of new increase concept \( C_{new} \).

**Theorem 2** in concept lattice \( L(K) \), if \( C_1 = (SC_1, PC_1) \) and \( C = (SC, PC) \) are generation sub concept of new increase concept \( C_{new} = (Extension(C) \cap SC_1, Intension(C) \cup PC_1) \).
**Theorem 3** supposed in formerly concept lattice \( L(K_i) \) and \( L(K_j) \), concept ordered by extension’s power from little to great. If concept \( C_i \) of concept lattice \( L(K_i) \) is update concept or increase concept of concept \( C \) in correspond \( L(K_j) \), concept \( C' \) of \( L(K_j) \) insert into \( L(K_i) \) after concept \( C \), thus does not need consider concept operation between \( C' \) and concept \( C_i \).

**IV. Algorithm**

Portrait Union Algorithm of Multiple Concept Lattices pseudo-code description as following:

**INPUT:** concept lattice \( L(K_i) \) and concept \( (SC, PC) \)

**OUTPUT:** new concept lattice \( L(K_{new}) \)

**BEGIN**

FOR each concept node \( (SC_i, PC_i) \) in \( L(K_i) \) sort ascending by \( |SC_i| \)

IF node \( (SC_i, PC_i) \) update THEN CONTINUE ENDIF

IF \( SC_i \subseteq SC \) THEN /*update concept*/

Add \( PC \) to \( PC_i \), \( PC_i = PC_i \cup PC \);

Insert \( (SC_i, PC_i) \) into VISITED.CS

Set \( (SC_i, PC_i) \) node update or new add tag

IF \( SC_i = SC \) THEN exit ENDIF

ELSE

\( SC_{new} = SC_i - SC \); /*generation sub concept*/

IF does not exist some \( (SC_i', PC_i') \) in VISITED.CS which satisfies \( SC_i' = SC_{new} \) THEN creates a new node \( C_{new} = (SC_{new}, PC_i \cup PC) \);

Add edge \( (SC_i, PC_i) \rightarrow C_{new} \):

FOR each node \( C_a \) in VISITED.CS DO

IF \( Extension(C_a) = SC_{new} \) THEN

Child := true;

FOR \( C_a \)'s every parent node \( C_p \) DO

IF \( Extension(C_p) = SC_{new} \) THEN child := false;

**END**

**ENDFOR**

**END**

Break;

ENDIF

ENDFOR

IF child THEN

IF \( C_a \) is child node of \( (SC_i, PC_i) \) THEN delete edge \( (SC_i, PC_i) \rightarrow C_a \)

ENDIF

ENDIF

ENDFOR

Add \( C_{new} \) into VISITED.CS

Set \( C_{new} \) node update tag

ENDIF

ENDFOR

END

Give a demonstration of the algorithm.

Divide formal context shown in table1 into two sub context \( O_1 = (S_1, P_1, R_1) \) and \( O_2 = (S_2, P_2, R_2) \), where \( P_1 = \{TP_1, TP_2\} \) and \( P_2 = \{TP_3, TP_4\} \). Corresponding sub lattice are \( L(O_1) \) and \( L(O_2) \), respectively shown in figure 2 and figure 3.

![Fig. 2 formal context corresponded the concept lattice L(O1)](image1)

![Fig. 3 formal context corresponded the concept lattice L(O2)](image2)

Now add nodes of \( L(O_2) \) to \( L(O_1) \) in turn.

**Step 1** add node \#1: First compute \#1, \{\( \cup \{TP_1, TP_4\}\}\), \{\( TP_2, TP_3\)\}, gain node \#5\{\( \cup \{TP_1, TP_2, TP_3, TP_4\}\), in \( L(O_1) \) and \( L(O_2) \) there does not exist node less than \#1 and \#1', so node \#5\{\( \cup \{TP_1, TP_2, TP_3, TP_4\}\) is new increase node, its generator sub formula is \#1; \#1' and \#1's next node generation extension
like #5, thus does not generate new increase node, after add #1', lattice shown in figure 4, where overstriking node is new increase node, overstriking real line denotes of connection between it and its generator sub node.

Step 2 add node #2': it adds after node #1' adds, thus does not consider operation between it and new increase node #5. Compute with node #1', because $\{TS_1,TS_4\}\cap\{TS_2,TS_3\}=$\{\}, does not generate new node, compute #2 $\{TS_1,TS_2,TS_4,TS_5\}\cap\{TS_2,TS_3\}=$\{$TS_2\}$, $\{TP\}\cap\{TP_3\}=$\{$TP_1,TP_3\}$, gain node #6($\{TS_2\}$, $\{TP_1,TP_3\}$). In $L(O_1)$ and $L(O_2)$ nodes less than #2 and less than #2' have not equal or more than #6, so node #6($\{TS_2\}$, $\{TP_1,TP_3\}$) is a new increase node. Its generate sub formula is #2', by the same way, compute with node #3 and generate new increase node #7($\{TS_3\}$, $\{TP_1,TP_3\}$). Compute with node #4 and generate new increase node #8($\{TS_2,TS_3\}$, $\{TP_3\}$), here new increase nodes which express add to $L(O_1)$, after add #2' lattice shown in figure 5.

V. Conclusions

In this paper, we have invested the problem of constructing and merging ontology of Web service using concept lattice theory. We form concept lattice and have a novel view to solution the service ontology constructing. And then we conclude the study as a Portrait Union of Multiple Concept lattices algorithm based on the discussion before. A merge example has illustrated the algorithm. Our future work is constructing actual service ontology based on concept and extending the proposed results to another research problem.

VI. Acknowledgment

The research is sponsored by Shaanxi Provincial Department of Education Project (11JK1042).

References


