Ontology Merging for Federated Ontologies on the Semantic Web

Gerd Stumme,[1] Alexander Maedche[2]

[1] Institute for Applied Informatics and Formal Description Methods (AIFB)
University of Karlsruhe, D-76128 Karlsruhe, Germany;
www.aifb.uni-karlsruhe.de/WBS/gst
[2] FZI Research Center for Information Technologies
Haid-und-Neu-Strasse 10-14, D–76131 Karlsruhe, Germany; www.fzi.de/wim

Abstract. One of the core challenges for the Semantic Web is the aspect of decentralization. Local structures can be modeled by ontologies. However, in order to support global communication and knowledge exchange, mechanisms have to be developed for integrating the local systems. We adopt the database approach of autonomous federated database systems and consider an architecture for federated ontologies for the Semantic Web as starting point of our work.
We identify the need for merging specific ontologies for developing federated, but still autonomous web systems. We present the method FCA–MERGE for merging ontologies following a bottom-up approach which offers a structural description of the merging process. The method is guided by application-specific instances of the given source ontologies that are to be merged. We apply techniques from natural language processing and formal concept analysis to derive a lattice of concepts as a structural result of FCA–MERGE. The generated result is then explored and transformed into the merged ontology with human interaction.

1 Introduction

The current WWW is a great success with respect to the amount of stored documents and the number of users. One of the main reasons for the success of the current WWW is the principle of *decentralization* [Be99]. Currently the Semantic Web, developed as a "metaweb" for the WWW, is being established by standards for syntax (e. g. XML) and semantics (RDF(S), DAML+OIL, etc.). Ontologies have been established for knowledge sharing and are widely used as a means for conceptually structuring domains of interest. One of the core challenges for the Semantic Web is the aspect of decentralization.[1] Local structures can be modeled by ontologies. However, in order to support global communication and knowledge exchange, mechanisms have to be developed for integrating the local systems.

[1] cf. http://www.w3.org/DesignIssues/Principles.html

A number of proposals are available from the database community for developing multi-database systems and, more specific, federated database systems, that resemble the decentralized structures required in the Semantic Web. We adopt the database approach of federated databases and consider an architecture for federated ontologies on the Semantic Web as motivation and starting point of our work.

A bottleneck for federated ontologies in the Semantic Web is the process of integrating or merging specific ontologies. The process of *ontology merging* takes as input two (or more) source ontologies and returns a merged ontology based on the given source ontologies. Manual ontology merging using conventional editing tools without support is difficult, labor intensive and error prone. Therefore, several systems and frameworks for supporting the knowledge engineer in the ontology merging task have recently been proposed [Ho98,Ch00,NM00,MFRW00]. The approaches rely on syntactic and semantic matching heuristics which are derived from the behavior of ontology engineers when confronted with the task of merging ontologies, i. e. human behaviour is simulated. Although some of them locally use different kinds of logics for comparisons, these approaches do not offer a structural description of the global merging process.

We propose the new method FCA–Merge for merging ontologies following a bottom-up approach which offers a global structural description of the merging process. For the source ontologies, it extracts instances from a given set of domain-specific text documents by applying natural language processing techniques. Based on the extracted instances we apply mathematically founded techniques taken from *Formal Concept Analysis* [Wi82,GW99] to derive a lattice of concepts as a structural result of FCA–Merge. The produced result is explored and transformed to the merged ontology by the ontology engineer. The extraction of instances from text documents circumvents the problem that in most applications there are no objects which are simultaneously instances of the source ontologies, and which could be used as a basis for identifying similar concepts.

The remainder of the paper is as follows. We start our paper introducing a generic architecture for federating ontologies for the Semantic Web in Section 2. There we also identify the need for merging specific ontologies for developing federated, autonomous systems. We briefly introduce some basic definitions concentrating on a formal definition of what an ontology is and recall the basics of Formal Concept Analysis in Section 3. In Sections 4 to 6, we present our method FCA–Merge for merging ontologies following a bottom-up approach which offers a global structural description of the merging process. We present our generic method for ontology merging in Section 4. Section 5 provides a detailed description of FCA–Merge. Section 6 gives an overview over related work, and Section 7 summarizes the paper and concludes with an outlook on future work.

2 An Architecture for Federated Ontologies in the Semantic Web

Figure 1 depicts the 5–layer architecture of federated ontologies on the Semantic Web. It adopts the approach of [SL90] for federated databases.

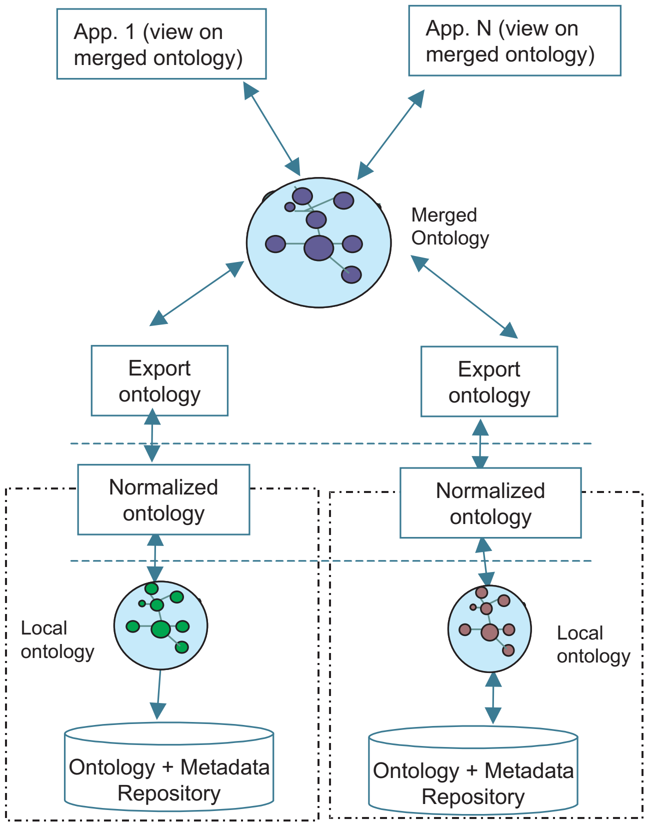

Fig. 1. Architecture for Federated Ontologies

The architecture extends the standardized 3–layer schema architecture ANSI/SPARC with two additional layers. The adopted architecture mainly consists of:

1. local ontologies (the conceptual models of the autonomous systems), each of them with its specific underlying ontology/metadata repository or database,
2. normalized ontologies (transformation of the local ontologies into a common data model),
3. export ontologies (view on the normalized ontology that describes the relevant parts of the ontology for the federation),

4. one merged ontology (global ontology derived from the combination of the two export schemas), and
5. different applications in the upper layer (external schema layer), which use the merged ontology with their specific views on it.

In the following we will not go into further details of the organizational and architectural structure. As already mentioned, the following sections and the rest of this paper are dedicated to the task of generating a merged ontology from the two (or more) given export ontologies of the autonomous web systems.

3 Ontologies and Formal Concept Analysis

In this section, we briefly introduce some basic definitions. We thereby concentrate on a formal definition of what an ontology is and recall the basics of Formal Concept Analysis.

3.1 Ontologies

There is no common formal definition of what an ontology is. However, most approaches share a few core items: concepts, a hierarchical IS-A-relation, and further relations. For sake of generality, we do not discuss more specific features like constraints, functions, or axioms here. We formalize the core in the following way.

Definition: A *(core) ontology* is a tuple $\mathcal{O} := (\mathcal{C}, \texttt{is\_a}, \mathcal{R}, \sigma)$, where $\mathcal{C}$ is a set whose elements are called *concepts*, $\texttt{is\_a}$ is a partial order on $\mathcal{C}$ (i. e., a binary relation $\texttt{is\_a} \subseteq \mathcal{C} \times \mathcal{C}$ which is reflexive, transitive, and anti-symmetric), $\mathcal{R}$ is a set whose elements are called *relation names* (or *relations* for short), and $\sigma\colon \mathcal{R} \to \mathcal{C}^+$ is a function which assigns to each relation name its arity.

As said above, the definition considers the core elements of most languages for ontology representation only. It is possible to map the definition to most types of ontology representation languages. Our implementation, for instance, is based on Frame Logic [KLW95]. Frame Logic has a well-founded semantics, but we do not refer to it in this paper.

3.2 Formal Concept Analysis

We recall the basics of Formal Concept Analysis (FCA) as far as they are needed for this paper. A more extensive overview is given in [GW99]. To allow a mathematical description of concepts as being composed of extensions and intensions, FCA starts with a *formal context* defined as a triple $\mathbb{K} := (G, M, I)$, where G is a set of *objects*, M is a set of *attributes*, and I is a binary relation between G and M (i. e. $I \subseteq G \times M$). $(g, m) \in I$ is read "*object g has attribute m*".

Definition: For $A \subseteq G$, we define $A' := \{m \in M \mid \forall g \in A\colon (g, m) \in I\}$ and, for $B \subseteq M$, we define $B' := \{g \in G \mid \forall m \in B\colon (g, m) \in I\}$.

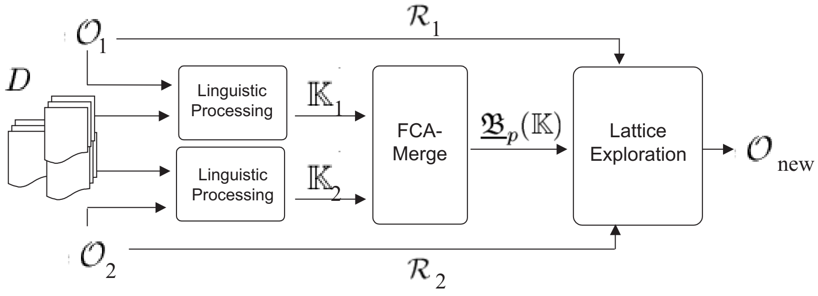

Fig. 2. Ontology Merging Method

A *formal concept* of a formal context (G, M, I) is defined as a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$. The sets A and B are called the *extent* and the *intent* of the formal concept (A, B). The *subconcept–superconcept relation* is formalized by $(A_1, B_1) \leq (A_2, B_2) :\Longleftrightarrow A_1 \subseteq A_2 \quad (\Longleftrightarrow B_1 \supseteq B_2)$. The set of all formal concepts of a context $\mathbb{K}$ together with the partial order $\leq$ is always a complete lattice,[2] called the *concept lattice* of $\mathbb{K}$ and denoted by $\underline{\mathfrak{B}}(\mathbb{K})$.

A possible confusion might arise from the double use of the word ‘concept’ in FCA and in ontologies. This comes from the fact that FCA and ontologies are two models for the concept of ‘concept’ which arose independently. In order to distinguish both notions, *we will always refer to the FCA concepts as ‘formal concepts’. The concepts in ontologies are referred to just as ‘concepts’ or as ‘ontology concepts’.* There is no direct counter-part of formal concepts in ontologies. Ontology concepts are best compared to FCA attributes, as both can be considered as unary predicates on the set of objects.

4 Bottom-Up Ontology Merging

As said above, we propose a bottom-up approach for ontology merging. Our mechanism is based on application-specific instances of the two given ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ that are to be merged. The overall process of merging two[3] ontologies is depicted in Figure 2 and consists of three steps, namely *(i)* instance extraction and computing of two formal contexts $\mathbb{K}_1$ and $\mathbb{K}_2$, *(ii)* the FCA-Merge core algorithm that derives a common context and computes a concept lattice, and *(iii)* the generation of the final merged ontology based on the concept lattice.

Our method takes as input data the two ontologies and a set D of natural language documents. The documents have to be relevant to both ontologies, so

[2] I. e., for each set of formal concepts, there is always a greatest common subconcept and a least common superconcept.

[3] The approach can easily be extended for merging n instead of two ontologies simultaneously.

that the documents are described by the concepts contained in the ontology. The documents may be taken from the target application which requires the final merged ontology. From the documents in D, we *extract instances*. The mechanism for instance extraction is further described in Subsection 5.1. This automatic knowledge acquisition step returns, for each ontology, a formal context indicating which ontology concepts appear in which documents.

The extraction of the instances from documents is necessary because there are usually no instances which are already classified by both ontologies. However, if this situation is given, one can skip the first step and use the classification of the instances directly as input for the two formal contexts.

The second step of our ontology merging approach comprises the FCA–Merge core algorithm. The core algorithm merges the two contexts and computes a concept lattice from the merged context using FCA techniques. More precisely, it computes a *pruned concept lattice* which has the same degree of detail as the two source ontologies. The techniques applied for generating the pruned concept lattice are described in Subsection 5.2 in more detail.

Instance extraction and the FCA–Merge core algorithm are fully automatic. The final step of *deriving the merged ontology* from the concept lattice requires human interaction. Based on the pruned concept lattice and the sets of relation names $\mathcal{R}_1$ and $\mathcal{R}_2$, the ontology engineer creates the concepts and relations of the target ontology. We offer graphical means of the ontology engineering environment OntoEdit for supporting this process.

For obtaining good results, a few assumptions have to be met by the input data: Firstly, the documents have to be relevant to each of the source ontologies. A document from which no instance is extracted for each source ontology can be neglected for our task. Secondly, the documents have to cover all concepts from the source ontologies. Concepts which are not covered have to be treated manually after our merging procedure (or the set of documents has to be expanded). And last but not least, the documents must separate the concepts well enough. If two concepts which are considered as different always appear in the same documents, FCA-Merge will map them to the same concept in the target ontology (unless this decision is overruled by the knowledge engineer). When this situation appears too often, the knowledge engineer might want to add more documents which further separate the concepts.

5 The FCA–Merge Method

In this section, we discuss the three steps of FCA–Merge in more detail. We illustrate FCA–Merge with a small example taken from the tourism domain, where we have built several specific ontology-based information systems. Our general experiments are based on tourism ontologies that have been modeled in an ontology engineering seminar. Different ontologies have been modeled for a given text corpus on the web, which is provided by a WWW provider for tourist information.[4] The corpus describes actual objects, like locations, accommoda-

[4] URL: http://www.all-in-all.com

tions, furnishings of accommodations, administrative information, and cultural events. For the scenario described here, we have selected two ontologies: The first ontology contains 67 concepts and 31 relations, and the second ontology contains 51 concepts and 22 relations. The underlying text corpus consists of 233 natural language documents taken from the WWW provider described above. For demonstration purposes, we restrict ourselves first to two very small subsets $\mathcal{O}_1$ and $\mathcal{O}_2$ of the two ontologies described above; and to 14 out of the 233 documents. These examples will be translated in English. In Subsection 5.3, we provide some examples from the merging of the larger ontologies.

5.1 Linguistic Analysis and Context Generation

The aim of this first step is to generate, for each ontology $\mathcal{O}_i, i\in\{1,2\}$, a formal context $\mathbb{K}_i := (G_i, M_i, I_i)$. The set of documents D is taken as object set ($G_i := D$), and the set of concepts is taken as attribute set ($M_i := \mathcal{C}_i$). While these sets come for free, the difficult step is generating the binary relation I_i. The relation $(g,m) \in I_i$ shall hold whenever document g contains an instance of m.

The computation uses linguistic techniques as described in the sequel. We conceive an information extraction-based approach for ontology-based extraction, which has been implemented on top of SMES (Saarbrücken Message Extraction System), a shallow text processor for German (cf. [NBB+97]). The architecture of SMES comprises a *tokenizer* based on regular expressions, a *lexical analysis* component including a *word and a domain lexicon*, and a *chunk parser*. The tokenizer scans the text in order to identify boundaries of words and complex expressions like "$20.00" or "Mecklenburg–Vorpommern",[5] and to expand abbreviations.

The lexicon contains more than 120,000 stem entries and more than 12,000 subcategorization frames describing information used for lexical analysis and chunk parsing. Furthermore, the domain-specific part of the lexicon contains lexical entries that express natural language representations of concepts and relations. Lexical entries may refer to several concepts or relations, and one concept or relation may be referred to by several lexical entries.

Lexical analysis uses the lexicon to perform *(1)* morphological analysis, i. e. the identification of the canonical common stem of a set of related word forms and the analysis of compounds, *(2)* recognition of named entities, *(3)* part-of-speech tagging, and *(4)* retrieval of domain-specific information. While steps (1), (2), and (3) can be viewed as standard for information extraction approaches, step (4) is of specific interest for our instance extraction mechanism. This step associates single words or complex expressions with a concept from the ontology if a corresponding entry in the domain-specific part of the lexicon exists. For instance, the expression "Hotel Schwarzer Adler" is associated with the concept `Hotel`. If the concept `Hotel` is in ontology $\mathcal{O}_1$ and document g contains the expression "Hotel Schwarzer Adler", then the relation $(g,\texttt{Hotel})\in I_1$ holds.

[5] a region in the north east of Germany

I_1	Vacation	Hotel	Event	Concert	Root
doc1	×	×	×	×	×
doc2	×	×	×	×	×
doc3	×	×	×		×
doc4	×	×	×	×	×
doc5			×	×	×
doc6		×	×	×	×
doc7		×			×
doc8	×	×	×	×	×
doc9	×	×	×		×
doc10	×	×	×		×
doc11	×	×	×	×	×
doc12		×			×
doc13		×	×	×	×
doc14	×	×	×		×

I_2	Hotel	Accommodation	Musical	Root
doc1	×	×	×	×
doc2	×	×	×	×
doc3	×	×		×
doc4	×	×	×	×
doc5			×	×
doc6	×	×	×	×
doc7	×	×		×
doc8	×	×	×	×
doc9	×	×		×
doc10	×	×		×
doc11	×	×	×	×
doc12	×	×		×
doc13	×	×	×	×
doc14	×	×		×

Fig. 3. The contexts $\mathbb{K}_1$ and $\mathbb{K}_2$ as result of the first step

Finally, the transitivity of the `is_a`-relation is compiled into the formal context, i. e. $(g,m) \in I$ and m `is_a` n implies $(g,n) \in I$. This means that if $(g,$`Hotel`$) \in I_1$ holds and `Hotel is_a Accommodation`, then the document also describes an instance of the concept `Accommodation`: $(g,$`Accommodation`$) \in I_1$.

Figure 3 depicts the contexts $\mathbb{K}_1$ and $\mathbb{K}_2$ that have been generated from the documents for the small example ontologies. E. g., document `doc5` contains instances of the concepts `Event`, `Concert`, and `Root` of ontology $\mathcal{O}_1$, and `Musical` and `Root` of ontology $\mathcal{O}_2$. All other documents contain some information on hotels, as they contain instances of the concept `Hotel` both in $\mathcal{O}_1$ and in $\mathcal{O}_2$.

5.2 Generating the Pruned Concept Lattice

The second step takes as input the two formal contexts $\mathbb{K}_1$ and $\mathbb{K}_2$ which were generated in the last step, and returns a *pruned concept lattice* (see below), which will be used as input in the next step.

First we merge the two formal contexts into a new formal context $\mathbb{K}$, from which we will derive the pruned concept lattice. Before merging the two formal contexts, we have to disambiguate the attribute sets, since $\mathcal{C}_1$ and $\mathcal{C}_2$ may contain the same concepts: Let $\widetilde{M_i} := \{(m,i) \mid m \in M_i\}$, for $i \in \{1,2\}$. The indexation of the concepts allows the possibility that the same concept exists in both ontologies, but is treated differently. For instance, a `Campground` may be

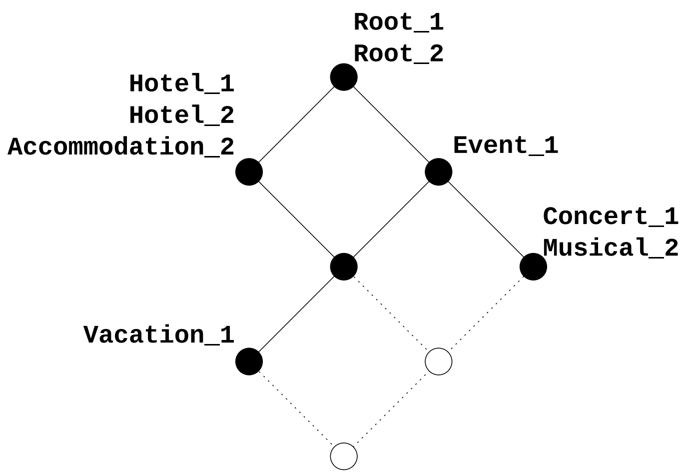

Fig. 4. The pruned concept lattice

considered as an `Accommodation` in the first ontology, but not in the second one. Then the merged formal context is obtained by $\mathbb{K} := (G, M, I)$ with $G := D$, $M := \widetilde{M_1} \cup \widetilde{M_2}$, and $(g, (m, i)) \in I :\Leftrightarrow (g, m) \in I_i$.

We will not compute the whole concept lattice of $\mathbb{K}$, as it would provide too many too specific concepts. We restrict the computation to those formal concepts which are above at least one formal concept generated by an (ontology) concept of the source ontologies. This assures that we remain within the range of specificity of the source ontologies. More precisely, the *pruned concept lattice* is given by $\underline{\mathfrak{B}}_p(\mathbb{K}) := \{(A, B) \in \underline{\mathfrak{B}}(\mathbb{K}) \mid \exists m \in M \colon (\{m\}', \{m\}'') \leq (A, B)\}$ (with $\cdot'$ as defined in Section 3.2).

For our example, the pruned concept lattice is shown in Figure 4. It consists of six formal concepts. Two formal concepts of the total concept lattice are pruned since they are too specific compared to the two source ontologies. In the diagram, each formal concept is represented by a node. The empty nodes are the pruned concepts and are usually hidden from the user. A concept is a subconcept of another one if and only if it can be reached by a descending path. The intent of a formal concept consists of all attributes (i. e., in our application, the ontology concepts) which are attached to the formal concept or to one of its superconcepts. As we are not interested in the document names, the extents of the contexts are not visualized in this diagram.

The computation of the pruned concept lattice is done with the algorithm Titanic [STB+00]. It is modified to allow the pruning. The modified algorithm is described below.

Compared to other algorithms for computing concept lattices, Titanic has — for our purpose — the advantage that it computes the formal concepts via their *key sets* (or *minimal generators*). A key set is a minimal description of a formal concept:

Definition 1. *$K \subseteq M$ is a key set for the formal concept (A, B) if and only if $(K', K'') = (A, B)$ and $(X', X'') \neq (A, B)$ for all $X \subseteq K$ with $X \neq K$.*[6]

In our application, key sets serve two purposes. Firstly, they indicate if the generated formal concept gives rise to a new concept in the target ontology or not. A concept is new if and only if it has no key sets of cardinality one. Secondly, the key sets of cardinality two or more can be used as generic names for new concepts and they indicate the arity of new relations.

The Titanic Algorithm. We recall the algorithm TITANIC and discuss how it is modified to compute the pruned concept lattice. In the following, we will use the composed function $\cdot'' : \mathfrak{P}(M) \to \mathfrak{P}(M)$ which is a closure operator on M (i. e., it is extensive, monotonous, and idempotent). The related closure system (i. e., the set of all $B \subseteq M$ with $B'' = B$) is exactly the set of the intents of all concepts of the context. The structure of the concept lattice is already determined by this closure system. Hence we restrict ourselves to the computation of all concept intents in the sequel. The computation makes extensive use of the following *support* function:

Definition 2. *The* support *of $X \subseteq M$ is defined by $s(X) := \frac{|X'|}{|G|}$.*

We follow a pruning strategy given in [AS94]. Originally this strategy was presented as a heuristic for determining all frequent sets only (i. e., all sets with supports above a user-defined threshold). The algorithm traverses the powerset of M in a level-wise manner. At the kth iteration, all subsets of M with cardinality k (called *k-sets*) are considered, unless we know in advance that they cannot be key sets.

The pseudo-code of the modified TITANIC algorithm is given in Algorithm 1. A list of notations is provided in Table 1.

Table 1. Notations used in TITANIC

k	is the counter which indicates the current iteration. In the kth iteration, all key k-sets are determined.
$\mathcal{K}_k$	contains after the kth iteration all key k-sets K together with their weight $K.s$ and their closure K.closure.
$\mathcal{C}$	stores the candidate k-sets C together with a counter $C.p\_s$ which stores the minimum of the weights of all $(k-1)$-subsets of C. The counter is used in step 8 to prune all non-key sets.

[6] In other words: K generates the formal concept (A, B).

Algorithm 1 TITANIC

1) $\emptyset.s \leftarrow 1$;
2) $\mathcal{K}_0 \leftarrow \{\emptyset\}$;
3) $k \leftarrow 1$;
4) **forall** $m \in M$ **do** $\{m\}.p\_s \leftarrow 1$;
5) $\mathcal{C} \leftarrow \{\{m\} \mid m \in M\}$;
6) **loop begin**
7) COUNT($\mathcal{C}$);
8) $\mathcal{K}_k \leftarrow \{X \in \mathcal{C} \mid X.s \neq X.p\_s$ and $(k = 1$ or $\exists m \in M\colon X \subseteq m.\text{closure})\}$;
9) **forall** $X \in \mathcal{K}_k$ **do** X.closure $\leftarrow$ CLOSURE(X);
10) **if** $\mathcal{K}_k = \emptyset$ **then exit loop** ;
11) $k++$;
12) $\mathcal{C} \leftarrow$ TITANIC-GEN($\mathcal{K}_{k-1}$);
13) **end loop** ;
14) **return** $\bigcup_{i=0}^{k-1}\{X.\text{closure} \mid X \in \mathcal{K}_i\}$.

The algorithm starts with stating that the empty set is always a key set, and that its support is always equal to 1 (steps 1+2). Then all 1-sets are candidate sets by definition (steps 4+5). In later iterations, the candidate k-sets are determined by the function TITANIC-GEN (step 12/Algorithm 2) which is (except step 5) equivalent to the generating function of Apriori. (The result of step 5 will be used in step 8 of Algorithm 1 for pruning the non-key sets.)

Once the candidate k-sets are determined, the function COUNT($\mathcal{X}$) is called to compute, for each $X \in \mathcal{X}$, the support of X. It is stored in the variable $X.s$ (step 7).

In step 8 of Algorithm 1, the second condition prunes all candidate k-sets which are out of the range of the two source ontologies. I. e., it implements the condition of the definition of the pruned concept lattice $\underline{\mathfrak{B}}_p(\mathbb{K})$. This additional condition makes the difference to the algorithm presented in [STB+00]. The first condition in step 8 prunes all candidate k-sets which are not key sets according to Proposition 1.

Proposition 1 ([STB+00]). *$X \subseteq M$ is a key set if and only if $s(X) \neq \min_{m \in X}(s(X \setminus \{m\}))$.*

For the remaining sets (which are now known to be key sets) their closures are computed (step 9). The CLOSURE function (Algorithm 3) is a straight-forward implementation of Proposition 2 (beside an additional optimization (step 2)).

Proposition 2 ([STB+00]).

1. *Let $X \subseteq M$. Then $h(X) = X \cup \{m \in M \setminus X \mid s(X) = s(X \cup \{m\})\}$.*
2. *If X is not a key set, then $s(X) = \min\{s(K) \mid K \in \mathcal{K}, K \subseteq X\}$ where $\mathcal{K}$ is the set of all key sets.*

Algorithm 1 terminates, if there are no key k-sets left (step 10+14). Otherwise the next iteration begins (steps 11+12).

Algorithm 2 TITANIC-GEN

We assume that there is a total order $>$ on M.

Input: $\mathcal{K}_{k-1}$, the set of key $(k-1)$-sets K with their support $K.s$.

Output: $\mathcal{C}$, the set of candidate k-sets C
with the values $C.p\_s := \min\{s(C \setminus \{m\} \mid m \in C\}$.

The variables p_s assigned to the sets $\{p_1, \ldots, p_k\}$ which are generated in step 1 are initialized by $\{p_1, \ldots, p_k\}.p\_s \leftarrow 1$.

1) $\mathcal{C} \leftarrow \{\{p_1, \ldots, p_k\} \mid i < j \Rightarrow p_i < p_j,\ \{p_1, \ldots, p_{k-2}, p_{k-1}\},\ \{p_1, \ldots, p_{k-2}, p_k\} \in \mathcal{K}_{k-1}\}$;
2) **forall** $X \in \mathcal{C}$ **do begin**
3)   **forall** $(k-1)$-subsets S of X **do begin**
4)   **if** $S \notin \mathcal{K}_{k-1}$ **then begin** $\mathcal{C} \leftarrow \mathcal{C} \setminus \{X\}$; **exit forall** ; **end**;
5) $X.p\_s \leftarrow \min(X.p\_s, S.s)$;
6)   **end**;
7) **end**;
8) **return** $\mathcal{C}$.

Algorithm 3 CLOSURE(X) for $X \in \mathcal{K}_{k-1}$

$Y \leftarrow X$;
forall $m \in X$ **do** $Y \leftarrow Y \cup (X \setminus \{m\})$.closure;
forall $m \in M \setminus Y$ **do begin**
    if $X \cup \{m\} \in \mathcal{C}$ **then** $s \leftarrow (X \cup \{m\}).s$
            else $s \leftarrow \min\{K.s \mid K \in \mathcal{K},\ K \subseteq X \cup \{m\}\}$;
    if $s = X.s$ **then** $Y \leftarrow Y \cup \{m\}$
end;
return Y.

5.3 Generating the new Ontology from the Concept Lattice

While the previous steps (instance extraction, context derivation, context merging, and TITANIC) are fully automatic, the derivation of the merged ontology from the concept lattice requires human interaction, since it heavily relies on background knowledge of the domain expert.

The result from the last step is a pruned concept lattice. From it we have to derive the target ontology. Each of the formal concepts of the pruned concept lattice is a candidate for a concept, a relation, or a new subsumption in the target ontology. There is a number of queries which may be used to focus on the most relevant parts of the pruned concept lattice. We discuss these queries after the description of the general strategy — which follows now. Of course, most of the technical details are hidden from the user.

As the documents are not needed for the generation of the target ontology, we restrict our attention to the intents of the formal concepts, which are sets of (ontology) concepts of the source ontologies. For each formal concept of the pruned concept lattice, we analyze the related key sets. For each formal concept, the following cases can be distinguished:

1. It has exactly one key set of cardinality 1.
2. It has two or more key sets of cardinality 1.
3. It has no key sets of cardinality 0 or 1.
4. It has the empty set as key set.[7]

The generation of the target ontology starts with all concepts being in one of the two first situations. The first case is the easiest: The formal concept is generated by exactly one ontology concept from one of the source ontologies. It can be included in the target ontology without interaction of the knowledge engineer. In our example, these are the two formal concepts labeled by `Vacation_1` and by `Event_1`.

In the second case, two or more concepts of the source ontologies generate the same formal concept. This indicates that the concepts should be merged into one concept in the target ontology. The user is asked which of the names to retain. In the example, this is the case for two formal concepts: The key sets {`Concert_1`} and {`Musical_2`} generate the same formal concept, and are thus suggested to be merged; and the key sets {`Hotel_1`}, {`Hotel_2`}, and {`Accommodation_2`} also generate the same formal concept.[8] The latter case is interesting, since it includes two concepts of the same ontology. This means that the set of documents does not provide enough details to separate these two concepts. Either the knowledge engineer decides to merge the concepts (for instance because he observes that the distinction is of no importance in the target application), or he adds them as separate concepts to the target ontology. If there are too many suggestions to merge concepts which should be distinguished, this is an indication that the set of documents was not large enough.[9] In such a case, the user might want to re-launch FCA–MERGE with a larger set of documents.

When all formal concepts in the first two cases are dealt with, then all concepts from the source ontologies are included in the target ontology. Now, all relations from the two source ontologies are copied into the target ontology. Possible conflicts and duplicates have to be resolved by the ontology engineer.

In the next step, we deal with all formal concepts covered by the third case. They are all generated by at least two concepts from the source ontologies, and are candidates for new ontology concepts or relations in the target ontology. The decision whether to add a concept or a relation to the target ontology (or to discard the suggestion) is a modeling decision, and is left to the user. The key sets provide suggestions either for the name of the new concept, or for the concepts which should be linked with the new relation. Only those key sets with

[7] This implies (by the definition of key sets) that the formal concept does not have another key set.

[8] {`Root_1`} and {`Root_2`} are no key sets, as each of them has a subset (namely the empty set) generating the same formal concept.

[9] The same holds for suggested subsumptions. This is for instance the case for the concept `Vacation_1`, which is always mentioned in the documents whenever `Hotel_1` is mentioned, and which is thus suggested to become a subconcept of the latter.

minimal cardinality are considered, as they provide the shortest names for new concepts and minimal arities for new relations, resp.

For instance, the formal concept in the middle of Figure 4 has {Hotel_2, Event_1}, {Hotel_1, Event_1}, and {Accommodation_2, Event_1} as key sets. The user can now decide if to create a new concept with the default name HotelEvent (which is unlikely in this situation), or to create a new relation with arity (Hotel, Event), e. g., the relation organizesEvent.

Key sets of cardinality 2 serve yet another purpose: $\{m_1, m_2\}$ being a key set implies that neither m_1 is_a m_2 nor m_2 is_a m_1 currently hold. Thus when the user does not use a key set of cardinality 2 for generating a new concept or relation, she should check if it is reasonable to add one of the two subsumptions to the target ontology. This case does not show up in our small example. An example from the large ontologies is given at the end of the section.

There is exactly one formal concept in the fourth case (as the empty set is always a key set). This formal concept gives rise to a new largest concept in the target ontology, the Root concept. It is up to the knowledge engineer to accept or to reject this concept. Many ontology tools require the existence of such a largest concept. In our example, this is the formal concept labeled by Root_1 and Root_2.

Finally, the is_a order on the concepts of the target ontology can be derived automatically from the pruned concept lattice: If the concepts c_1 and c_2 are derived from the formal concepts (A_1, B_1) and (A_2, B_2), resp., then c_1 is_a c_2 if and only if $B_1 \supseteq B_2$ (or if explicitly modeled by the user based on a key set of cardinality 2).

Querying the pruned concept lattice. In order to support the knowledge engineer in the different steps, there is a number of queries for focusing his attention to the significant parts of the pruned concept lattice.

Two queries support the handling of the second case (in which different ontology concepts generate the same formal concept). The first is a list of all pairs $(m_1, m_2) \in \mathcal{C}_1 \times \mathcal{C}_2$ with $\{m_1\}' = \{m_2\}'$. It indicates which concepts from the different source ontologies should be merged.

In our small example, this list contains for instance the pair (Concert_1, Musical_2). In the larger application (which is based on the German language), pairs like (Zoo_1, Tierpark_2) and (Zoo_1, Tiergarten_2) are listed. We decided to merge Zoo [engl.: zoo] and Tierpark [zoo], but not Zoo and Tiergarten [zoological garden].

The second query returns, for ontology $\mathcal{O}_i$ with $i \in \{1, 2\}$, the list of pairs $(m_i, n_i) \in \mathcal{C}_i \times \mathcal{C}_i$ with $\{m_i\}' = \{n_i\}'$. It helps checking which concepts out of a single ontology might be subject to merge. The user might either conclude that some of these concept pairs can be merged because their differentiation is not necessary in the target application; or he might decide that the set of documents must be extended because it does not differentiate the concepts enough.

In the small example, the list for $\mathcal{O}_1$ contains only the pair (Hotel_1, Accommodation_1). In the larger application, we had additionally pairs like (Räumliches, Gebiet) and (Auto, Fortbewegungsmittel). For the target application, we merged `Räumliches` [spatial thing] and `Gebiet` [region], but not `Auto` [car] and `Fortbewegungsmittel` [means of travel].

The number of suggestions provided for the third situation can be quite high. There are three queries which present only the most significant formal concepts out of the pruned concepts. These queries can also be combined.

Firstly, one can fix an upper bound for the cardinality of the key sets. The lower the bound is, the fewer new concepts are presented. A typical value is 2, which allows to retain all concepts from the two source ontologies (as they are generated by key sets of cardinality 1), and to discover new binary relations between concepts from the different source ontologies, but no relations of higher arity. If one is interested in having exactly the old concepts and relations in the target ontology, and no suggestions for new concepts and relations, then the upper bound for the key set size is set to 1.

Secondly, one can fix a minimum support. This prunes all formal concepts where the cardinality of the extent is too low (compared to the overall number of documents). In Algorithm 1, this is achieved by adding the condition "[...] and $X.s \geq$ minsupp" to step 8. The default is no pruning, i. e., with a minimum support of 0 %. It is also possible to fix different minimum supports for different cardinalities of the key sets. The typical case is to set the minimum support to 0 % for key sets of cardinality 1, and to a higher percentage for key sets of higher cardinality. This way we retain all concepts from the source ontologies, and generate new concepts and relations only if they have a certain (statistical) significance.

Thirdly, one can consider only those key sets of cardinality 2 in which the two concepts come from one ontology each. This way, only those formal concepts are presented which give rise to concepts or relations linking the two source ontologies. This restriction is useful whenever the quality of each source ontology *per se* is known to be high, i. e., when there is no need to extend each of the source ontologies alone.

In the small example, there are no key sets with cardinality 3 or higher. The three key sets with cardinality 2 (as given above) all have a support of $\frac{11}{14} \approx 78.6\,\%$. In the larger application, we fixed 2 as upper bound for the cardinality of the key sets. We obtained key sets like (`Telefon_1` [telephone], `Öffentliche_Einrichtung_2` [public institution]) (support = 24.5 %), (`Unterkunft_1` [accommodation], `Fortbewegungsmittel_2` [means of travel]) (1.7 %), (`Schloß_1` [castle], `Bauwerk_2` [building]) (2.1 %), and (`Zimmer_1` [room], `Bibliothek_2` [library]) (2.1 %). The first gave rise to a new concept `Telefonzelle` [public phone], the second to a new binary relation `hatVerkehrsanbindung` [hasPublicTransportConnection], the third to a new subsumption `Schloß is_a Bauwerk`, and the fourth was discarded as meaningless.

6 Related Work

A first approach for supporting the merging of ontologies is described in [Ho98]. There, several heuristics are described for identifying corresponding concepts in different ontologies, e. g. comparing the names and the natural language definitions of two concepts, and checking the closeness of two concepts in the concept hierarchy.

The OntoMorph system [Ch00] offers two kinds of mechanisms for translating and merging ontologies: syntactic rewriting supports the translation between two different knowledge representation languages, semantic rewriting offers means for inference-based transformations. It explicitly allows to violate the preservation of semantics in trade-off for a more flexible transformation mechanism.

In [MFRW00] the Chimaera system is described. It provides support for merging of ontological terms from different sources, for checking the coverage and correctness of ontologies and for maintaining ontologies over time. Chimaera offers a broad collection of functions, but the underlying assumptions about structural properties of the ontologies at hand are not made explicit.

Prompt [NM00] is an algorithm for ontology merging and alignment embedded in Protg 2000. It starts with the identification of matching class names. Based on this initial step an iterative approach is carried out for performing automatic updates, finding resulting conflicts, and making suggestions to remove these conflicts.

The tools described above offer extensive merging functionalities, most of them based on syntactic and semantic matching heuristics, which are derived from the behaviour of ontology engineers when confronted with the task of merging ontologies. OntoMorph and Chimarea use a description logics based approach that influences the merging process locally, e. g. checking subsumption relationships between terms. None of these approaches offers a structural description of the global merging process. FCA–Merge can be regarded as complementary to existing work, offering a structural description of the overall merging process with an underlying mathematical framework.

There is also much related work in the database community, especially in the area of federated database systems. The work closest to our approach is described in [SS98] and [Co97]. They apply Formal Concept Analysis to a related problem, namely database schema integration. As in our approach, a knowledge engineer has to interpret the results in order to make modeling decisions. Our technique differs in two points: There is no need of knowledge acquisition from a domain expert in the preprocessing phase; and it additionally suggests new concepts and relations for the target ontology.

7 Conclusion and Future Work

We have motivated our work with the issue of decentralization, one of the main challenges for the Semantic Web. We have adopted the database point of view and consider an architecture for federating ontologies in the Semantic Web as motivation of our work. We discussed especially the process of integrating or merging specific ontologies which is a bottleneck for federated ontologies in the Semantic Web.

In this paper we have presented FCA–Merge, a bottom-up technique for merging ontologies based on a set of documents. We have described the three steps of the technique: the linguistic analysis of the texts which returns two formal contexts; the merging of the two contexts and the computation of the pruned concept lattice; and the semi-automatic ontology creation phase which supports the user in modeling the target ontology. The paper described the underlying assumptions and discussed the methodology.

Future work includes the closer integration of the FCA–Merge method in the ontology engineering environment OntoEdit. In particular, we will offer views on the pruned concept lattice based on the queries described in Subsection 5.3. It is also planned to further refine our information-extraction based mechanism for extracting instances. This refinement goes hand in hand with further improvements concerning the connection between ontologies and natural language (cf. [MSS+01]).

The evaluation of ontology merging is an open issue [NM00]. We plan to use FCA–Merge to generate independently a set of merged ontologies (based on two given source ontologies). Comparing these merged ontologies using the standard information retrieval measures as proposed in [NM00] will allow us to evaluate the performance of FCA–Merge.

On the theoretical side, an interesting open question is the extension of the formalism to features of specific ontology languages, like for instance functions or axioms. The question is (i) how they can be exploited for the merging process, and (ii) how new functions and axioms describing the interplay between the source ontologies can be generated for the target ontology.

Future work also includes the implementation of the framework of federated ontologies as introduced in Section 2. We refer the interested reader to the recently started EU-IST funded project OntoLogging[10], where the development and management of federated web systems consisting of multiple ontologies and associated knowledge bases will be studied and implemented.

Acknowledgements

This research was partially supported by DFG and BMBF.

References

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proc. VLDB Conf.*, 1994, 478–499 (Expanded version in IBM Report RJ9839)

[Ch00] H. Chalupsky: OntoMorph: A translation system for symbolic knowledge. *Proc. 7th Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR'2000)*, Breckenridge, Colorado, USA, April 2000, 471–482

[Co97] S. Conrad: *Föderierte Datenbanksysteme: Konzepte der Datenintegration.* Informatik-Lehrbuch, Springer, Berlin–Heidelberg 1997

[10] http://www.ontologging.com

[GW99] B. Ganter, R. Wille: *Formal Concept Analysis: mathematical foundations.* Springer, Berlin–Heidelberg 1999

[Be99] T. Berners-Lee: *Weaving the Web.* Harper, Berlin–Heidelberg 1999

[Ho98] E. Hovy: Combining and standardizing large-scale, practical ontologies for machine translation and other uses. *Proc. 1st Intl. Conf. on Language Resources and Evaluation*, Granada, Spain, May 1998.

[KLW95] M. Kifer, G. Lausen, J. Wu: Logical foundations of object-oriented and frame-based languages. *Journal of the ACM* **42**(4) 1995, 741–843

[MSS+01] A. Maedche, S. Staab, N. Stojanovic, R. Studer, Y. Sure: SEAL - A Framework for Developing SEmantic Web PortALs. In: Brian J. Read (ed.): *Advances in Databases*, Proc. 18th British National Conference on Databases, BNCOD 18, LNCS **2097**, Springer, Heidelberg 2001, 1–22

[MFRW00] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder: An environment for merging and testing large Oontologies. *Proc. 7th Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR'2000)*, Breckenridge, Colorado, USA, April 2000, 483–493.

[NBB+97] G. Neumann, R. Backofen, J. Baur, M. Becker, C. Braun: An information extraction core system for real world German text processing. *Proc. ANLP-97*, Washington, USA, 1997

[NM00] N. Fridman Noy, M. A. Musen: PROMPT: algorithm and tool for automated ontology merging and alignment. *Proc. 17th Natl. Conf. on Artificial Intelligence (AAAI'2000)*, Austin, TX, July/August 2000, 450–455

[SS98] I. Schmitt, G. Saake: Merging inheritance hierarchies for database integration. *Proc. 3rd Int. Conf. on Cooperative Information Systems (CoopIS'98)*, IEEE Computer Science Press 1998, 322–331.

[STB+00] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal: Fast computation of concept lattices using data mining techniques. *Proc. KRDB '00*, CEUR-Workshop Proc. http://CEUR-WS.org/Vol-29, 129–139.

[SL90] A. Sheth, J. Larsen: Federated database systems for managing distributed, heterogeneous and autonomous databases. *ACM Computing Surveys*, **22**(3), 1990

[Wi82] R. Wille: Restructuring lattice theory: an approach based on hierarchies of concepts. In: I. Rival (ed.): *Ordered sets.* Reidel, Dordrecht–Boston 1982, 445–470.