Comparison of «Personal Ontologies» Represented through Conceptual Graphs

Rose Dieng[1] and Stefan Hug[2]

Abstract. When modeling expertise from several experts, comparing their different expertise models can help to build their common expertise model. The viewpoint of an expert on the domain can be considered as his/her «personal ontology». Such a personal ontology can be represented using Sowa's conceptual graph formalism. This paper presents an algorithm aimed at helping a knowledge engineer to build the common ontology of several experts from their personal ontologies, when such ontologies are represented through Sowa's conceptual graph formalism.

1 INTRODUCTION

1.1 Building ontologies

Among the numerous definitions of ontologies [9], an ontology is often defined as an *«explicit knowledge level specification of a conceptualization»* [24]. To represent a given phenomenon or a given domain, it is necessary to restrict to a number of significant concepts sufficient to interpret this domain [23]. An ontology relies on some ontological commitments (that guide the choice of a given set of concepts rather than another). An ontology comprises definitions providing a vocabulary allowing to communicate about a domain. It allows to define (a) the terms usable to describe knowledge, (b) relations among such terms and (c) their use constraints. The form of definitions depends on the language chosen for representing them at the symbol level: for example, KIF in the Ontolingua server [8] [6], the language CML offered by CommonKADS method [18] and conceptual graph formalism [21] allow to describe ontologies. Ontologies are particularly interesting when they describe the consensual terminology shared by a community. Several approaches support a cooperative building of a consensual ontology: CO4 protocol [5] allows several users to build a consensual knowledge base cooperatively, Ontolingua server allows distributed collaborative ontology construction [6], APECKS server [22] allows experts to compare their «personal ontologies» cooperatively.

1.2 Representing ontologies through conceptual graphs

Our approach tries to automate some parts of the process of building consensual knowledge. The viewpoint of an expert on a domain can be considered as his/her personal ontology [22]. Such an ontology can be described through Sowa's conceptual graph formalism [21].

Let us recall briefly the model of simple conceptual graphs, as defined in [21] [2]. A *support* S is a tuple = ($\mathcal{T}$c, $\mathcal{T}$r, $\mathcal{B}$, $\mathcal{M}$, conf) where $\mathcal{T}$c is a *lattice of concept types*, $\mathcal{T}$r is a *partially ordered set of relation types* ($\mathcal{T}$r may also have a structure of lattice), $\mathcal{B}$ is a set of "star graphs" in bijection with $\mathcal{T}$r, and indicating the signature of each relation type, $\mathcal{M}$ is a set of *individual markers* allowing to name and distinguish the individuals (i.e. instances), conf is a *conformity relation*, relating concept type labels to individual markers (i.e. it allows to specify if an individual marker is conform to a given type). In addition, we will suppose that to each concept type or relation type t, the function Name(t) associates its main name and the function Syn(t) its set of synonyms. A *conceptual graph defined with respect to a support S* is a connected, bipartite, labelled graph, the labelling respecting some constraints. A *base of canonical graphs* allows to represent the meaningful graphs for the considered domain.

So, the notion of support is useful to represent an ontology: the concept type lattice allows to describe the main concepts of the domain; the relation type hierarchy allows to describe the main relations among them. Definitions (i.e. necessary and sufficient conditions) and schemata associated to concept types or relation types are useful to describe constraints of use of such types. Moreover, the possibility to indicate main name and synonyms, and to associate definitions to types is useful for terminological purposes.

1.3 Our approach

Searching the common support associated to several experts of the same domain can be included in the process of searching a common ontology or a shared ontology among the experts. One can work either at the knowledge level, without choosing a representation formalism or at the symbol level, once chosen a representation formalism. Our choice of the framework of the conceptual graph formalism allows us to propose algorithms based on the notions underlying conceptual graphs. Of course, we don't hope to automate all the steps necessary for building such a common support from multiple experts : clearly, this construction must be cooperative and involve the experts in order to solve some conflicts.

In [3], we proposed a conflict management mechanism based on the comparison and integration of conceptual graphs representing knowledge of multiple experts. We rely on this previous related work. We associate to an expert an artificial agent, having (1) a support (made of a concept type lattice, a relation type lattice, and a set of markers satisfying a conformity relation w.r.t. the concept type lattice) representing the view of this expert on the world, and (2) a base of canonical conceptual graphs, built on this support and representing his/her expertise.

In this paper, we suppose that the personal ontology of an expert can be represented through such a support. We detail our techniques for comparison of supports of two agents (identification of the types of two hierarchies, fusion of hierarchies), before discussing the interest and drawbacks of our approach.

[1] INRIA, Centre de Sophia-Antipolis, Acacia Project, 2004 route des Lucioles, BP 93, 06902 Sophia-Antipolis Cedex, FRANCE

[2] PTV Planungsbüro Transport und Verkehr GmbH, Stumpfstraße 1, 76131 Karlsruhe, GERMANY

ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd

2 COMPARISON OF TWO SUPPORTS

The building of the common support from the supports of two experts relies on the following steps:

1. *Comparison and fusion of the concept type lattices* of the two experts: it must allow to solve name conflicts, and, in case of need, to add new types or to adapt type definitions.
2. *Comparison and fusion of the relation type lattices* used by the two experts: it must allow to solve name conflicts, and, in case of need, to add new types, to adapt type definitions and to adapt relation signatures.
3. *Comparison and fusion of the two sets of markers*: it must help to solve name conflicts and to adapt the conformity relation.

2.1 Identification of two types in hierarchies

The identification algorithm aims at determining, in the two concept type lattices or in the two relation type lattices, which types are identical. Let $Hier_1$ be a hierarchy of $Expert_1$ (i.e. the lattice without $\perp$) and let $Hier_2$ be the hierarchy of same nature of $Expert_2$. The identification algorithm relies on two phases:

1. *Phase 1: terminology-based identification.* This phase relies on the names to compare any type of $Hier_1$ and any type of $Hier_2$.
2. *Phase 2: context-based identification.* This phase relies on the contexts of two types, i.e. their relatives such as direct supertypes and direct subtypes in the hierarchies, in order to find whether they are identical.

2.1.1. First phase: terminology-based identification

In this first phase, the algorithm tries to identify which types of both hierarchies are «similar», according to their main names and their synonyms. Let t_1 be a type of $Hier_1$ and t_2 a type of $Hier_2$. Several identification criteria can be thought out and combined by the knowledge engineer (KE):

1. t_1 and t_2 have the same main name: Name (t_1) = Name (t_2).
2. The number of common synonyms of t1 and t_2 (i.e. $|\,Syn(t_1) \cap Syn(t_2)\,|^{*}$) is greater than a given threshold. (*We note |E| the cardinal of a finite set E.)
3. The main name of one type belongs to the set of synonyms of the other type:
   $Name(t_1) \in Syn(t_2) \vee Name(t_2) \in Syn(t_1)$.

To each criterion a function : $Hier_1 * Hier_2 \rightarrow N$ and a weight (a real number) are associated, as summarized in Table 1. A *similarity function* $Sim_1 : Hier_1 * Hier_2 \rightarrow R$ calculates the similarity factor between a type of $Hier_1$ and a type of $Hier_2$, according to this first identification phase.

$Sim_1(t_1, t_2)$ = SameMainName(t_1,t_2) * RMainName
+ NumberOfSameSynonyms(t_1,t_2) * RSynonyms
+ MainNameInSynonyms(t_1,t_2) * RMainInSyn.

Let $n = |Hier_1|$ and $m = |Hier_2|$ where $|Hier_i|$ is the number of concept types in $Hier_i$. The types of $Hier_1$ are numbered from 1 to n and the types of $Hier_2$ are numbered from 1 to m. The results of the similarity function are stored by the identification algorithm in a similarity matrix SimMatr of n lines and m columns. If t_1 is the type numbered i in $Hier_1$ and t_2 the type numbered j in $Hier_2$, the ith line and jth column of this matrix contains the value $SimMatr_{ij} = Sim_1(t_1, t_2)$. Once the matrix filled with all the values $SimMatr_{ij}$, the algorithm considers the highest value obtained in each row and checks whether this value is ≥ to a threshold $T_{similar}$.

Table 1: Identification functions

Function	Value	Weight
SameMainName(t_1,t_2)	* 1 if Name(t_1) = Name (t_2) * 0 otherwise	RMainName
NumberOfSameSynonyms(t_1,t_2)	$\mid Syn(t_1) \cap Syn(t_2) \mid$	RSynonyms
MainNameInSynonyms(t_1,t_2)	* 1 if $(Name(t_1) \in Syn(t_2)) \vee (Name(t_2) \in Syn(t_1))$ * 0 otherwise	RMainInSyn

Definition: After this first phase, two types t_1 and t_2 are *«1-similar»* iff $Sim_1(t_1,t_2)$ $T_{similar}$. We also note 1-similar (t_1,t_2).

Remark: For a given type t_1, there may be several types t_2 found 1-similar with t_1. But the second phase will help to discriminate them more thoroughly.

2.1.2. Second phase : context-based identification

Once established the similarity matrix after the first phase, the identification algorithm then considers the contexts of the types to be compared. The context of a type is described through its relatives, i.e. its direct supertypes (predecessors), and its direct subtypes (successors) in the type hierarchy. In this second phase, the algorithm tries to identify which types of both hierarchies are the same, according to their contexts.

For $i \in \{1,2\}$, let t_i be a type of $Hier_i$, let Pred (t_i) be the set of direct supertypes of t_i in $Hier_i$ and let Succ (t_i) be the set of direct subtypes of ti in $Hier_i$.

$SamePred(t_i, t_j) = \{e_i \in Pred(t_i) ; \exists e_j \in Pred(t_j) : \text{1-similar}(e_i, e_j)\}$
$SameSucc(t_i, t_j) = \{e_i \in Succ(t_i) ; \exists e_j \in Succ(t_j) : \text{1-similar}(e_i, e_j)\}$

Three identification cases are distinguished:

1. *Case 1:* The number of 1-similar direct supertypes (resp. subtypes) of t_1 and t_2 is ≥ to a threshold Tpred (resp. Tsucc) :
   $(|SamePred(t_1, t_2)| \geq Tpred) \wedge (|SameSucc(t_1,t_2)| \geq Tsucc)$
2. *Case 2:* All the predecessors (resp. successors) of t_1 and t_2 are 1-similar:
   $|SamePred(t_1,t_2)| = |Pred(t_1)| = |Pred(t_2)| = |SamePred(t_2,t_1)|$
   $\wedge |SameSucc(t_1,t_2)| = |Succ(t_1)| = |Succ(t_2)| = |SameSucc(t_2,t_1)|$
3. *Case 3:* $\exists\, i, j \in \{1,2\}$ with $i \neq j$, the set of relatives of t_i is «included» in the set of relatives of t_j , w.r.t. 1-similarity:
   $(SameSucc(t_i,t_j) = Succ(t_i)) \wedge (|Succ(t_i)| \leq |Succ(t_j)|)$
   $\wedge (SamePred(t_i,t_j) = Pred(t_i)) \wedge (|Pred(t_i)| \leq |Pred(t_j)|)$

The weights associated to the three cases are respectively called RRelatives, RAllRelatives and RAllRelativesInc. They have a default value, but they can be tailored by the KE.

In the second phase, another similarity function $Sim_2 : Hier_1 * Hier_2 \rightarrow R$ is calculated :

$Sim_2(t_1, t_2)$ = RRelatives in case 1,
RAllRelatives in case 2
RAllRelativesInc in case 3.

The new factor $Sim_2(t_1, t_2)$ is added in the similarity matrix SimMatr. If t_1 is the type numbered i in $Hier_1$ and t_2 the type numbered j in $Hier_2$, $SimMatr_{ij} = Sim_1(t_1, t_2) + Sim_2(t_1, t_2)$

Then, the couples of identical types are calculated from the similarity matrix. The parameters and weights can be adapted by the KE.

Definition *After the second phase, the types t_1 (numbered i in $Hier_1$) and t2 (numbered j in $Hier_2$) are considered as identical iff:*
- $SimMatr_{ij}$ is the maximal value in the ith line in the matrix and in the jth column of the matrix,
- and is greater than or equal to a threshold T_{same}.

If the maximal value in the ith row (resp. jth column) of the similarity matrix SimMatr is not unique, it means that there are several candidates in $Hier_2$ to be identical with t_1. In this case, the KE may play with the different parameters by refining the thresholds or the weights and by running again the algorithm till only one type of $Hier_2$ remains identical to t_1.

Remarks:

1. This 2-phases-based identification algorithm is used for the concept (resp. relation) type lattice comparison. The first phase (terminology-based identification) is also used for the comparison of the sets of markers, as it does not exploit the hierarchy structure.
2. The editor offered by our tool allows the KE to tune the parameters RMainName, etc (she does not handle directly the similarity matrix) and to visualize the hierarchies with different grains (e.g. by hiding subhierarchies of some nodes).

2.2 Fusion of the concept type lattices

In order to perform an automatic fusion of the two concept type lattices, the set I_{CType} of couples of identical concept types must be found by comparison of Tc_1 and Tc_2. Two comparison strategies can be applied, implemented in the two following algorithms: *One-to-one match algorithm* and *Hierarchy-match algorithm*.

2.2.1. One-to-one match algorithm

For each cycle of comparison in the two previously described phases of identification, the algorithm compares each type of Tc_1 to each type of Tc_2. The complexity of this algorithm is O(m*n).

2.2.2. Hierarchy-match algorithm

This algorithm, inspired by [12], takes into account the hierarchical structure in its comparison strategy. It relies on a depth-first search in both hierarchies and it proceeds as follows: once two identical types have been found, then a search for further identifications in their subhierarchies is performed. In this algorithm, the thresholds $T_{similar}$ and T_{same} have the same values. In both previous phases, after each evaluation of a couple of types (tc_1, tc_2), the corresponding value $SimMatr_{ij}$ is compared to T_{same}. As soon as $SimMatr_{ij}$ T_{same}, then the couple (tc_1,tc_2) is included in the set I_{CType}.

2.2.3. Preconditions for fusion

Let A and B be two lattices, and $\leq_A$ and $\leq_B$ their associated partial ordering relations and $\|_A$ and $\|_B$ their associated incomparability relations. Let I = { (a,b) $\in$ A * B ; a = b}.

A can be integrated in B if the following condition is satisfied [1]:
$\forall(a_1,b_1) \in I, \forall(a_2,b_2) \in I, a_1 \leq_A a_2 \Leftrightarrow b_1 \leq_B b_2 \wedge a_1 \|_A a_2 \Leftrightarrow b_1 \|_B b_2$

If this fusion condition is satisfied, then a MacNeille completion [13] can be performed afterwards in order to complete this integration so as to obtain again a lattice. Therefore, in order to enable the fusion of Tc_1 and Tc_2 and to obtain a lattice Tc_{com}, the algoritm MakeUnionPossible(I_{CType}) checks the fusion precondition and, in case of need, eliminates from I_{CType} the couples responsible for its violation.

2.2.4. Fusion of concept type lattices

For making the fusion of both concept type lattices, the KE can initialize the set I_{CType} by indicating which types of both hierarchies are already known as exactly identical. The universal types and the absurd types must necessarily belong to this initial set of identical concept types. Then the identification algorithm is applied (either with a one-to-one match strategy or with a hierarchy match strategy). Then, before the fusion, the partial ordering relation of identical types are checked. The couples responsible for violation of the fusion precondition are eliminated from I_{TCtype}. Then the integrated hierarchy Tc_{com} is built, by representing each couple of identical types by a single type in Tc_{com}, and by adding the types appearing only in one hierarchy.

Algorithm CTypeFusion (Tc1, Tc2)

begin
InitIdentSet (I_{CType}) ; SimilaritytIdent (I_{CType})
MakeUnionPossible (I_{CType}) ; T_{ccom} = Union (I_{CType})
CompleteLattice (T_{ccom})
end

Definition *Let Tc_{com} be the final integrated concept type hierarchy obtained after fusion of Tc_1 and Tc_2. The function $ctype_{com} : Tc_1 \cup Tc_2 \rightarrow Tc_{com}$ associates to any concept type of the two compared hierarchies its image in Tc_{com}.*

If a type appears only in the hierarchy of one expert and cannot be identified to any type of the second expert, then it will be kept in the integrated hierarchy, with a prefix in its name indicating from which expert it comes from. If a type appears in both hierarchies of experts, the experts can choose its final main name stored in Tc_{com}. In all cases, the associated synonyms are also stored in Tc_{com}.

2.2.5. Possible conflicts

In addition to terminology conflicts, topology conflicts and conflicts specific to conceptual graph formalism must be processed.

In [19], Shaw and Gaines define several kinds of terminological situations between two experts: consensus (cf. same terms for the same concepts), correspondence (different terms - synonyms - for the same concept), conflict (same terms - homonyms - for different concepts), contrast (different terms for different concepts).

Terminology conflicts stem from the concept type couples that have been classified as identical because of context criteria, but that have different labels (i.e. Shaw and Gaines's correspondence). In this case, the experts can choose the main name with which the final concept type will be stored in the integrated hierarchy Tc_{com}.

If the two type hierarchies to be merged differ with respect to their partial order, it may cause *topology conflicts*, processed by the algorithm MakeUnionPossible (I).

Conflicts specific to conceptual graph formalism may arise from cross-references between the three support components: they may arise from the signatures linking relation types to concept types or from the conformity relation linking individual markers and concept types. An adaptation to the newly created support is needed.

Figure 1 shows an example of topology conflict (cf. the position of *Evaluation-Error* w.r.t. *Interpretation-Error*). For this example, Figure 2 shows the result obtained with the one-to-one match followed by a MacNeille completion (cf. creation of *IntermediateType*) while Figure 3 illustrates the redundancies produced by the hierarchy match in this case (cf. duplication of subhierarchy of *Evaluation-Error*). Figure 4 shows a non conflicting example.

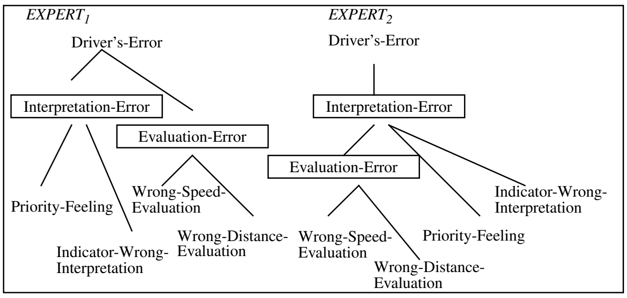

Figure 1. Example of topology conflict

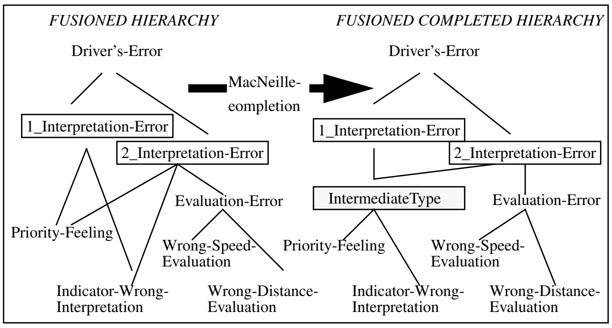

Figure 2. One-to-one Match with MacNeille completion

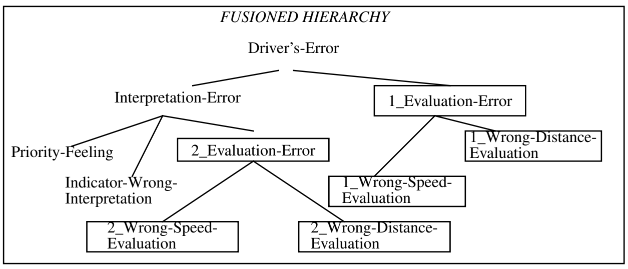

Figure 3. Hierarchy Match

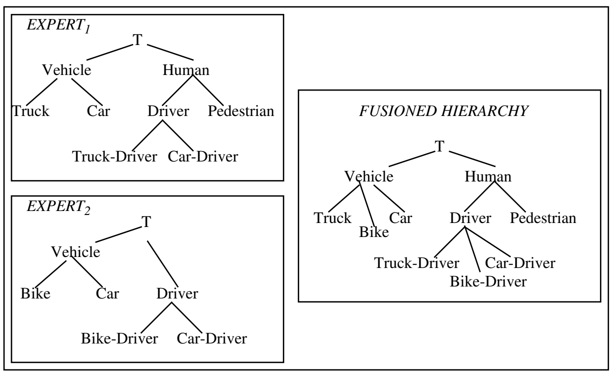

Figure 4. Example without topology conflict

2.3 Comparison and fusion of the relation type lattices

The identification algorithm for relation type hierarchies is similar to the two-phase-based algorithm previously presented. Once obtained the set I_{Rtype} of couples of identical relation types, the precondition for fusion of the two hierarchies must also be checked. Moreover, when two relation types are considered as identical, a verification of their signature compatibility must be performed. The signatures in the integrated relation type hierarchy Tr_{com} must be adapted according to the integrated concept type hierarchy Tc_{com}.

Algorithm RTypeFusion (Tr_1, Tr_2)

begin

InitIdentSet (I_{RType}) ; SimilarityIdent (I_{RType})
MakeUnionPossible (I_{RType}) ; AdaptSignatureForUniqueTypes (Tr_1)
AdaptSignatureForUniqueTypes (Tr_2)
foreach (tr_1, tr_2) $\in I_{RType}$ AdaptSignature (tr_1, tr_2) **end** ;
Tr_{com} = Union (I_{RType}) ; CompleteLattice (Tr_{com})

end

Definition *Let Tr_{com} be the integrated relation type hierarchy obtained from the fusion of Tr_1 and Tr_2. The function $rtype_{com} : Tr_1 \cup Tr_2 \rightarrow Tr_{com}$ associates to any relation type of the two compared hierarchies its image in the final integrated hierarchy Tr_{com}.*

For i $\in$ {1,2}, if tr_i is a n-ary relation type $\in Tr_i$, let us note its signature Sig (tr_i) = (tc_{i1}, ..., tc_{in}). The signature of a relation type $\in Tr_{com}$ satisfies:

- *Case 1:*
$\forall tr_1 \in Tr_1$ such that: $\neg \exists tr_2 \in Tr_2 : (tr_1, tr_2) \in I_{RType}$
Sig ($rtype_{com}$ (tr_1)) = ($ctype_{com}$ (tc_{11}), ..., $ctype_{com}$(tc_{1n}))
$\forall tr_2 \in Tr_2$ such that: $\neg \exists tr_1 \in Tr_1 : (tr_1, tr_2) \in I_{RType}$
Sig ($rtype_{com}$ (tr_2)) = ($ctype_{com}$ (tc_{21}), ..., $ctype_{com}$(tc_{2n}))
Informally, if a relation type appears only in one expert's hierarchy, its signature is «preserved» by the function $rtype_{com}$.
- *Case 2:*
$\forall tr_1 \in Tr_1$ such that: $\exists tr_2 \in Tr_2 : (tr_1, tr_2) \in I_{RType}$
$rtype_{com}$ (tr_1) = $rtype_{com}$ (tr_2) $\wedge$
Sig ($rtype_{com}$ (tr_1)) = ($ctype_{com}$(tc_{11}) $\cup$ $ctype_{com}$(tc_{21}),..., $ctype_{com}$(tc_{1n}) $\cup$ $ctype_{com}$(tc_{2n}))
The signature of the image of two identical relation types relies on the supremum of the concept types appearing in their signatures.

2.4 Comparison and fusion of the marker sets

The terminology-based identification algorithm is used for the set of markers. When two markers are identical, their conformity relation must be compatible, otherwise they are eliminated from I_{MType}.

Algorithm MarkerFusion (M_1, M_2)

begin

InitIdentList (I_{MType}) ; SimilarityIdent (I_{MType})
AdaptConformityForUniqueMarkers (M_1)
AdaptConformityForUniqueMarkers (M_2)
foreach (m_1, m_2) $\in I_{MType}$
if ($\neg$ AdaptConformity (m_1, m_2)) **then** RefuseIdentity (m_1, m_2) **fi end**
M_{com} = Union (I_{MType})

end

Definition *Let M_{com} be the integrated set of markers obtained from* *the fusion of M_1 and M_2. Then the function $mtype_{com} : M_1 \cup M_2 \rightarrow M_{com}$ associates to any individual marker its image in M_{com}.*

2.4.1. Conformity adaptation

For $i \in \{1,2\}$, for $m_i \in M_i$, let us note Min-tc $(m_i) \in Tc_i$ the minimal concept type to which the individual marker m_i is conform to. Then, for $m_{com} \in M_{com}$, its minimal conform concept type Min-tc $(m_{com}) \in M_{com}$ satisfies:

- *Case 1:*

$\forall\, m_1 \in M_1$ such that: $\neg \exists\, m_2 \in M_2 : (m_1, m_2) \in I_{MType}$
  Min-tc $(mtype_{com}(m_1))$ = $ctype_{com}$ (Min-tc (m_1))
$\forall\, m_2 \in M_2$ such that: $\neg \exists\, m_1 \in M_1 : (m_1, m_2) \in I_{MType}$
  Min-tc $(mtype_{com}(m_2))$ = $ctype_{com}$ (Min-tc (m_2))

Informally, if a marker is used by only one expert, $mtype_{com}$ «preserves» its minimal conform concept type.

- *Case 2*:

$\forall\, m_1 \in M_1$ such that: $\exists\, m_2 \in M_2 : (m_1, m_2) \in I_{MType}$
  Min-tc $(mtype_{com}(m_1))$ =
  $ctype_{com}$ (Min-tc (m_1)) $\cap$ $ctype_{com}$ (Min-tc (m_2))
$\Leftrightarrow$ ($ctype_{com}$ (Min-tc (m_1)) $\cap$ $ctype_{com}$ (Min-tc (m_2))) $\neq \perp$

The minimal concept type conform to the image of two identical markers is the infimum of their minimal conform concept types.

3 CONCLUSION AND DISCUSSION

In this paper, we proposed algorithms for comparison of supports of several experts, such supports representing for example their personal ontologies. The experts must of course take place in the process of integration and validate its results. The proposed algorithms were implemented in MULTIKAT [11], using C/C++, JAVA, and COGITO platform [10]. They were tested on personal ontologies of two experts in traffic accident analysis [4] (with around 90 concept types and 10 relation types in each support). Empirical results are detailed in [11]. Our algorithm could be improved by additional similarity functions taking into account special semantic dictionaries such as Wordnet dictionary [14] in the identification phase. Another possible improvement would be to exploit the comparison of conceptual graphs [3] in order to include a comparison of definitions or of schemata in the identification phase (as definitions and schemata are themselves represented by conceptual graphs).

We took inspiration of the study of terminology conflicts proposed in [19]. Our work can also be compared to research on integration of ontologies in the framework of conceptual graph formalism. This integration can be cooperative [7] or automatic as in [15], where it relies on a conceptual clustering technique, based on a syntactic similarity metrics between different term definitions. Our algorithm exploits the terminological and semantic information underlying the hierarchies handled in conceptual graph formalism, contrarily to the lattice embedding algorithm proposed in [17]. Guides for building shared ontologies are proposed in [20] and in [25], where the author also presents an algebra over ontologies, with a set of operations for matching and integrating ontologies. Other research in other formalisms is also interesting : semantic-based integration of conceptual schemata [16], comparison of classes of objects.

Compared with the cooperative building of a common ontology proposed in [22], in our approach, we try to integrate both automatic processing and cooperative building of the common ontology by relying on several hypotheses. One interest of our approach is the possible combination of several comparison criteria. But some problems remain to be solved: we need to guide the KE in the tailoring of the weights and thresholds exploited in the algorithm. Moreover, the initial hierarchies depend on the quality of the modelling performed by the KE. Last, the lattice constraint may be too strong: relaxing it could allow to avoid useless redundancy in the integrated hierarchy.

References

[1] G. Birkhoff. *Lattice Theory*, American Mathematical Society Colloquium Publications, vol. 25, 3rd edn., 1973.

[2] M. Chein and M. L.Mugnier. Conceptual Graphs: Fundamental Notions. *Revue d'Intelligence Artificielle.* **6**(4):365-406 , (1992).

[3] R. Dieng. Comparison of Conceptual Graphs for Modelling Knowledge of Multiple Experts. In Ras & al eds., *Foundations of Intelligent Systems, ISMIS'96*, Springer-Verlag, LNAI 1079, June 1996, p. 78-87.

[4] R. Dieng. *Comparison of Conceptual Graphs for Modelling Knowledge of Multiple Experts: Application to Traffic Accident Analysis*. INRIA Research Report RR-3161, April 1997.

[5] J. Euzenat. Corporate Memory through Cooperative Creation of Knowledge Based Systems and Hyper-Documents. In *Proc. of KAW'96*, p. 36-1 – 36-20, Banff, Canada, November 1996.

[6] A. Farquar, R. Fikes, J. Rice. The Ontolingua Server: a Tool for Collaborative Ontology Construction. In *Proc. of KAW'96*, p. 44-1 – 44-20, Banff, Canada, November 1996.

[7] C. Garcia. Co-Operative Building of an Ontology within Multi-Expertise Framework. *Proc. of COOP'96*, Juan-les-Pins, 1996, p. 435-454.

[8] T. Gruber A translation approach to portable ontology specifications. *Knowledge Acquisition*, **5**(2):199–220, (1993).

[9] N. Guarino Understanding, Building and Using Ontologies. *International Journal of Human-Computer Studies*, **46**:293-310, (1997).

[10] O. Haemmerlé. *CoGITo: une plate-forme de développement de logiciels sur les graphes conceptuels.* PhD Thesis, Univ. Montpellier, Jan. 1995.

[11] S. Hug. *Vergleich von Begriffsgraphen für die Modellierung von Wissen mehrerer Experten.* Master Thesis, Karlsruhe Univ., March 1997.

[12] K. Knight, S. Luk Building a large knowledge base for machine translation. *Proc. of AAAI'94*, Seattle, WA, 1994.

[13] H. M. MacNeille. Partially Ordered Sets. *Transactions of the American Mathematical Society*, **42**:416-460, (1936).

[14] G. A. Miller. WordNet a lexical database for English. *Communications of the ACM*, **38**(11):39 - 41, (1995).

[15] G. Mineau, M. Allouche. Establishing a Semantic Basis: Toward the Integration of Vocabularies. In *Proc. of the KAW'95*, p. 2-1 – 2-16, Banff, Canada, 1995.

[16] I. Mirbel. Semantic Integration of Conceptual Schemas. *Data and Knowledge Engineering,* **21**(2): 183-195, (1997).

[17] J. J. Sarbo. Lattice Embedding. In Eklund et al, eds, *Conceptual Structures: Knowledge Representation as Interlingua*, Springer-Verlag, LNAI 1115, pp. 293–307, Sydney, Australia, August 1996.

[18] G. Schreiber, B. Wielinga, H. Akkermans, W. van de Velde, A. Anjewierden. CML: The CommonKADS Conceptual Modelling Language. In Steels &al, eds, *A Future for Knowledge Acquisition: Proc. of EKAW'94*, pp. 1–25, Sept. 1994. Springer-Verlag, LNAI n. 867.

[19] M. L. G. Shaw and B. R. Gaines. A methodology for recognizing conflict, correspondence, consensus and contrast in a knowledge acquisition system. *Knowledge Acquisition*, **1**(4):341–363, (1989).

[20] D. Skuce. Conventions for Reaching Agreement on Shared Ontologies. In *Proc. of KAW'95*, p. 3-1 – 3-19, Banff, Canada,1995 .

[21] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Reading, Addison-Wesley, 1984.

[22] J. Tennison, N. Shadboldt. APECKS: a Tool to Support Living Ontologies. *Proc. of KAW'98*, Banff, Canada, April 1998.

[23] A. Valente, J. Breuker Towards Principled Core Ontologies. In *Proc. of KAW'96*, p. 33-1 – 33-20, Banff, Canada, 1996.

[24] G. Van Heijst, A. Schreiber, B. Wielinga Using Explicit Ontologies in KBS Development. *Int.Journ.Human.-Comp.Stud.*,**46**:183-298, (1997).

[25] G. Wiederhold. Interoperation, Mediation and Ontologies. *Proc. of FGCS'94 Workshop on Heterogeneous Cooperative Knowledge-Bases*, Tokyo, Japan, p.33-48, December 1994.