Assessing Heterogeneity by Classifying Ontology Mismatches

Pepijn R.S. Visser, Dean M. Jones, T.J.M. Bench-Capon and M.J.R. Shave

CORAL - Conceptualisation and Ontology Research at Liverpool
Department of Computer Science, University of Liverpool
PO Box 147, Liverpool, L69 7ZF, United Kingdom,
tel. (+44).151.794.3792, pepijn@csc.liv.ac.uk

Abstract. Efforts to integrate heterogeneous information sources face a trade off between interoperability and heterogeneity. Important integration obstacles arise from the differences in the underlying ontologies of the various sources. In this paper we investigate the impediments to integration, focussing on ontologies (rather than database schemes). In particular, we present a classification of ontology mismatches, and discuss how each of the mismatch types can be dealt with. The idea is that knowing which ontology mismatches are difficult to deal with may contribute to finding a balance between heterogeneity and interoperability.

1. Introduction[1]

Heterogeneity is both a welcome and an unwelcome feature for system designers. On the one hand heterogeneity is welcomed because it is closely related to system efficiency. The more a program can be tailored to the problem it is to solve, the more efficient it will be. Choosing programming languages and making ontological assumptions are examples of such 'tailoring' decisions (*e.g.,* [12]). On the other hand, heterogeneity in data and knowledge systems is considered an unwelcome feature because it proves to be an important obstacle for the interoperation of systems (*e.g.,* [7]; [15]). The lack of standards is an obstacle to the exchange of data between (existing) heterogeneous systems (*cf.* the legacy problem).

This dilemma illustrates the need to find a balance between heterogeneity and interoperability (*cf.* [14]). In this article we contribute to finding such a balance. We focus on heterogeneity at the ontological level ([8]). That is, we compare ontologies instead of representation language or database schemas. In particular, we study types of mismatches between two ontologies and the obstacles they present to system interoperation. The aim is to generate a collection of ontological-level heuristics that set out the boundaries of system heterogeneity and interoperability. Such heuristics could be beneficial in two respects. First, they would provide a means of assessing whether legacy systems can be integrated into an

[1] This article is a revised version of [17], an article that appeared in the Working Notes of the AAAI'97 Spring Symposium on Ontological Engineering at Stanford University, USA.

existing community of interoperating systems. Second, they would provide ontological guidance for the design of new systems that are to be integrated into an existing community.

The work presented in this article is part of the KRAFT (Knowledge Reuse and Fusion / Transformation) project. A multi-site research project on heterogeneous knowledge systems, conducted by the universities of Aberdeen, Cardiff, and Liverpool in cooperation with BT in the United Kingdom ([5])[2].

This remainder of this article is structured as follows. In section 2 we introduce some terminology that allows us to define ontology mismatches. Then, in section 3, we present a classification of ontology mismatches. In section 4 we discuss to what extent the ontology mismatches form an integration obstacle. In section 5 we discuss four issues that arose in comparing two example ontologies on the basis of the mismatch classification. Finally, in section 6 we draw some preliminary conclusions.

2. Terminology

Systems are heterogeneous if they have different characteristics. In determining whether systems are heterogeneous one can focus on different characteristics, yielding different types of heterogeneity. A commonly made distinction is that between *semantic heterogeneity* and *non-semantic* (or *syntactic*) *heterogeneity* (*e.g,* [10]). Here, we distinguish four types of heterogeneity. There is a *paradigm heterogeneity* if two systems express their knowledge using different modelling paradigms (*e.g.,* an object-oriented database and a relational database). There is a *language heterogeneity* if two systems express their knowledge in different representation languages (*e.g.,* horn-clause logic and production rules). There is an *ontology heterogeneity* if two systems make different ontological assumptions about their domain knowledge (*e.g.,* one system assuming houses to be composed of building material, such as bricks and windows, and another system assuming houses to consist of entities relevant for room rental, such as rooms, and showers). Finally, there is *content heterogeneity* if two systems represent different knowledge (we note that two systems that express the same knowledge but in different formalisms have a language heterogeneity, but not a content heterogeneity). The last two types, ontology and content heterogeneity, are instances of semantic heterogeneity, the first two types are instances of non-semantic heterogeneity. In this article we focus on systems with ontology heterogeneity (thus assuming an ontology to represent semantics).

Following Gruber in his original definition of an ontology that an ontology is an explicit specification of a conceptualisation ([6]) we assume the creation of an ontology to involve two sub processes; *conceptualising a domain*, and *explicating the conceptualisation* (this distinction forms the basis for our classification of ontology mismatches, the idea being that ontology mismatches may be introduced during both sub processes).

---

[2] More information on the KRAFT project can be found at: http://www.csc.liv.ac.uk/~pepijn/kraft.html

During the conceptualisation process decisions are made upon classes, instances, relations, functions and axioms that are distinguished in the domain. Usually, the process also involves ordering the classes in a hierarchical fashion, and assigning attributes to them. We assume the outcome of this process to be a conceptualisation. In particular, this conceptualisation consists of a set of ontology concept descriptions $\{C_1,..,C_n\}$ in which C_i is either a class description, an instance description, a relation description, a function description or an axiom description. In the conceptualisation process, we do not specify the form or the appearance of these descriptions as their specification is the topic of the explication process (although later on we will, for practical reasons, assume that it is a natural language expression).

Explicating the ontology-concept descriptions requires an ontology language. We do not choose a specific ontology language but confine ourselves to a generic form of an ontology. An ontology is defined as a 5-tuple $O = <CD, RD, FD, ID, AD>$ in which *CD* is a set of class definitions, *RD* is a set of relation definitions, *FD* is a set of function definitions, *ID* is a set of instance definitions, and *AD* is a set of axiom definitions[3]. In defining ontology mismatches we confine ourselves to mismatches between definitions of classes (*CD*), definitions of relations (*RD*), and definitions of instances (*ID*)[4]. Relations defined in *RD* are divided into relations that contribute to the taxonomic structure of the domain (*viz. structuring relations*), and those which do not (*viz. non-structuring relations*) ([9]). We consider definitions in *CD, RD,* and *ID* to be 3-tuples $Def = <T, D, C>$ in which *T* is the definiendum, *D* is the definiens (to avoid confusion with the definiens we use the letter *T* - for *term* - to denote the definiendum), and *C* is the ontology-concept description to be defined (distinguished during the conceptualisation process). The form C takes is of less importance here as long as we are able to determine whether two concept descriptions are identical (see below). For convenience, we here assume *C* to be expressed as a natural language expression (*cf.* the DOCUMENTATION relation which is linked to definitions in ONTOLINGUA). *T* is an atomic formula in a formal language and *D* is a (compound) formula in a formal language. Distinguishing the concept description to be defined (*C*) facilitates the expression of the intention of the knowledge engineer in making the definition, thereby allowing us to distinguish between different definitions with the same term and definiens(an example is given in section 3). An example of a definition (in PROLOG format) is the concept description 'A vessel is taken to be something large and seagoing' (*C*), which is explicated as vessel(X) ← seagoing(X) ∧ large(X), in which vessel(X) is the definiendum (*T*), and seagoing(X) ∧ large(X) is the definiens (*D*).

[3] These components are the components of an ontology defined in ONTOLINGUA, see [6].

[4] We do not address functions and axioms in our classification of ontology mismatches since their form is slightly different from the other definitions.

3. A classification of ontology mismatches

Semantic heterogeneity has been studied extensively in the database field (*e.g.,* [1], [13], [11], [2], [10]). Ceri and Widom ([2]), for instance, list four categories of semantic conflicts: (1) *naming conflicts* (different databases use different names to represent the same concepts), (2) *domain conflicts* (different databases use different values to represent the same concept), (3) *meta-data conflicts* (same concepts is represented at the schema level in one database and at the instance level in another database), and (4) *structural conflicts* (different databases use different data organisation to represent the same concept). Another area where semantic heterogeneity has been studied is knowledge acquisition. Shaw and Gaines ([16]), for instance, have proposed a method to compare and resolve conflicting conceptualisations of domain experts. Drawing on the distinction between concepts and the terminology to refer to concepts, they distinguish four different situations: (a) *consensus* (experts use terminology and concepts in the same way), (b) *correspondence* (experts differ in terminology but use the same concepts), (c) *conflict* (experts use same terminology for different concepts), and (d) *contrast* (experts differ in terminology and concepts).

The classification presented below (see also figure 1) is not necessarily restricted to knowledge acquisition or databases. Because we focus on ontologies the classification applies to databases as well as knowledge systems, and in fact, to every form of conceptualisation. The classifications mentioned above are subsumed by the classification of ontology mismatches that is presented here. We do not claim the classification to be complete nor do we claim the distinguished classes to be fully disjoint. The classification is thought to be a useful instrument to determine what mismatches are present between ontologies, to determine which mismatches are hard to resolve, and, to define guidelines for the design of interoperable systems. As stated before, there are two basic types of ontology mismatches: (*1*) *conceptualisation mismatches*, and (*2*) *explication mismatches*.

(*1*) *Conceptualisation mismatch.* Conceptualisation mismatches are mismatches between two (or more) conceptualisations of a domain. The conceptualisations differ in the ontological concepts distinguished or in the way these concepts are related.

(*1.1*) *Class mismatch.* A class mismatch is a conceptualisation mismatch relating to the classes distinguished in the conceptualisation. In particular, this type of mismatch concerns classes and their subclasses.

(*1.1.1*) *Categorisation mismatch.* A categorisation mismatch occurs when two conceptualisations distinguish the same class but divide this class into different subclasses. As an example we consider two conceptualisations that both contain knowledge about animals. A categorisation mismatch occurs when one conceptualisation structures its knowledge around classes mammals and birds, whereas the second structures its knowledge around classes carnivores and herbivores.

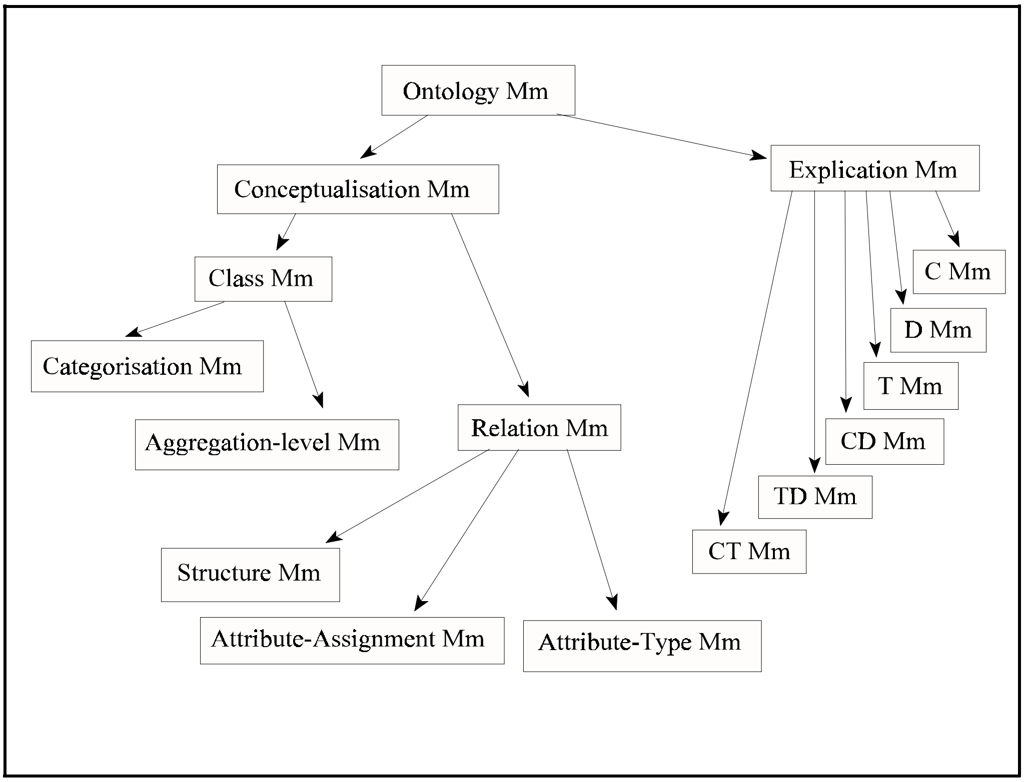

Figure 1: Classification of Ontology Mismatches

(*1.1.2*) *Aggregation-level mismatch.* An aggregation-level mismatch occurs if two conceptualisations both recognise the existence of a class, but define classes at different levels of abstraction. For instance, one conceptualisation that distinguished persons and a conceptualisation that distinguishes males and females but does not have persons as their superclass.

(*1.2*) *Relation mismatch.* A relation mismatch is a conceptualisation mismatch relating to the relations distinguished in the conceptualisation. Relation mismatches concern, for instance, the hierarchical relations between two classes or, the assignment of attributes to classes (*cf.* [19]).

(*1.2.1*) *Structure mismatch.* A structure mismatch occurs when two conceptualisations distinguish the same set of classes but differ in the way these classes are structured by means of relations. An example is one conceptualisation structuring classes house and brick with relation is-made-of, and another conceptualisation structuring the same classes with relation has-component. In this example, the two relations have a substantial overlap in their meaning but they are not equal; usually one does not say a house is made of a roof, whereas a roof can be a component of a house (the *dependency mismatch* in [1] is covered by this type of mismatch).

(*1.2.2*) *Attribute-assignment mismatch.* An attribute-assignment mismatch occurs when two conceptualisations differ in the way they assign an attribute (class) to other classes. Consider, for instance, two conceptualisations that both distinguish classes vehicle, car and colour in which car is a subclass of the class vehicle. The conceptualisations have an attribute-assignment mismatch if one conceptualisation assigns the attribute class colour to vehicle, and the other conceptualisation assigns colour to car.

(*1.2.3*) *Attribute-type mismatch.* An attribute-type mismatch occurs when two conceptualisations distinguish the same (attribute) class but differ in their assumed instantiations. An example of this type of mismatch is when two conceptualisations distinguishing the class length, assuming its instances to be a number of miles whereas another conceptualisation with the same class assumes its instances to be a number of kilometres.

(*2*) *Explication mismatch.* Explication mismatches are not defined on the conceptualisation of the domain but on the way the conceptualisation is specified. In section 2 we assumed that such an explicit conceptualisation (*viz.* an ontology) consists of a set of definitions. Recall that each definition in *CD*, *RD* and *ID* is a 3-tuple $Def = \langle T, D, C \rangle$ in which *T* is a term, *D* is a definiens, and *C* is the ontological concept (from the conceptualisation) that is explicated. The three components of a definition allow in principle for eight different relations between two definitions, and thus eight different types of mismatches. However, if terms, definiens, and concepts are all different we assume there to be no mismatch (*viz.* there is no correspondence). Neither is there a mismatch if terms, definiens, and concepts are all the same (*viz.* there is a complete match). This leaves six different types of mismatches. We assume an explication mismatch to occur when two ontologies have different definitions where their terms, their definiens, or their ontological concepts are identical.

(*2.1*) *CT mismatch* (or *Concept and Term mismatch*). A CT mismatch occurs when two ontologies use the same definiens *D* but differ in both the concept *C* they define and the term *T* linked to the definiens. For instance, if one ontology contains the definition vessel(X) ← seagoing(X) ∧ large(X) (to define the concept of a vessel) and the other ontology contains the definition whale(X) ← seagoing(X) ∧ large(X) (to define the concept of a whale).

(*2.2*) *CD mismatch* (or *Concept and Definiens mismatch*). A CD mismatch occurs when two ontologies use the same term *T* but differ in the concept *C* they define and the definiens *D* used for the definition. Consider the term mitre. One ontology may define the concept of the headgear of a bishop with the expression: mitre(X) ← head_dress_of_bishop(X), whereas a second ontology may define the concept of a straight angle joint of wood with the expression: mitre(X) ← straight_angle_joint_of_wood(X). Although both ontologies use the same term *T* it is clear that the ontologies denote a distinct concept and use a different definiens. Note, that a CD mismatch implies that *T* is a homonym.

(*2.3*) *C mismatch* (or *Concept mismatch*). A C mismatch occurs when both ontologies have the same term *T* and definiens *D* but differ in the concept they define. For instance, both ontologies could use the expression red_mitre(X) ← mitre(X) ∧ red(X) while one ontology defines the concept of a red piece of wood whereas the other ontology defines the concept of a red hat. A C mismatch is considered an explication mismatch rather than a conceptualisation mismatch since it can only be detected as such during the explication phase. Note that, like the CD mismatch, a C mismatch implies that *T* is a homonym.

(*2.4*) *TD mismatch* (or *Term and Definiens mismatch*). A TD mismatch occurs when two ontologies define the same concept *C* but differ in the way they define it; both with respect to the term *T* and the definiens *D*. For instance, two ontologies that define the concept of a ship with expressions, vessel(X) ← floating(X) ∧ big(X) and ship(X) ← seagoing(X) ∧ large(X), respectively.

Although representing the same concept in the world, these ontologies differ both in their term *T* and their definiens *D*. Note, that the TD mismatch implies that the two terms are synonyms (possibly the two definiens contain synonyms as well).

(*2.5*) *T mismatch* (or *Term mismatch*)[5]. A T mismatch occurs when two ontologies define the same concept *C* using the same definiens *D* but refer to it with different terms. Consider the use of the expression: $ship(X) \leftarrow seagoing(X) \wedge large(X)$ by one ontology and $vessel(X) \leftarrow seagoing(X) \wedge large(X)$ by another ontology, both defining the concept of a ship. Like the TD mismatch, the T mismatch implies that the two terms are synonyms.

(*2.6*) *D mismatch* (or *Definiens mismatch*). A D mismatch occurs when two ontologies define the same concept *C* and use the same term *T* to refer to the concept but use a different definiens. For instance, one ontology may define the concept of an ecclesiastical mitre with the expression: $mitre(X) \leftarrow head\_gear\_of\_bishop(X)$, whereas a second ontology may define the same term with the expression: $mitre(X) \leftarrow ecclesiastical\_hat(X) \wedge deeply\_cleft(X)$.

The two main categories of ontology mismatches reflect two different perspectives of looking at ontology mismatches. It should be noted that each conceptualisation mismatch is also present in the explication of that conceptualisation. Indeed, some conceptualisation mismatches are easily recognised as explication mismatches. Consider, for instance, the attribute-type mismatch. This conceptualisation mismatch type occurs in the explication as a D mismatch (or as a CD mismatch, depending on whether the type is specified in the description of the ontological concept C). In fact, all explication mismatches *must* occur in some form in the explication (ontology). However, not all explication mismatches necessarily occur in the conceptualisation. For instance, the actual terms (identifiers) to denote concepts are usually chosen in the explication process. This confirms the idea that certain ontological decisions are made only when the conceptualisation is explicated, and thus, that certain ontological mismatches occur only in the explication.

The reason why we adhere to both sets of ontology mismatches is twofold. First, it allows us to tell whether certain types of mismatches arise from the conceptualisation process, or from the explication process (which forms a basis to resolve them, see section 4). Second, some mismatches are better understood at a conceptual level (*viz.* in terms of classes and their hierarchical relations), whereas some mismatches are better understood in terms of ontology components (*viz.* in terms of terms and definiens). An example of the former mismatches is the categorisation mismatch which is clearly understood as a classes with different subclasses, but less obviously understood when expressed as a CD or D mismatch. An example of the latter mismatches is when one term refers to two different concepts (*viz.* a C or CD mismatch) which is clearly an explication mismatch, but maybe not recognised as a conceptualisation mismatch (for instance, because the terms are always used in their own context).

[5] Arguably, a T mismatch is no real ontological mismatch since it does not refer to a difference in conceptualisation. We here choose to refer to it as an ontology mismatch since an ontology is defined so as to include the terms that are used to refer to concepts. A mismatch in terms thus implies a mismatch in ontologies.

4. Ontological mismatches and interoperability

In finding a balance between heterogeneity and interoperability we have to examine to what extent the various ontology mismatches form an obstacle for interoperability. Having presented a classification of ontology mismatches, we now address the communication between two systems that have ontology mismatches. For each type of mismatches listed in the previous section we discuss whether we can define a mapping function that maps expressions based on one ontology onto expressions that are based on a second ontology (and *vice versa*). As such, the analysis provides some tentative requirements for adequate ontology mappings. For the mapping functions, we assume the formalisms in which the expressions are stated to be the same.

The scope of this article does not allow us to elaborate on all mismatch types and their resolution extensively, so all discussions are necessarily kept brief. Where possible we have included examples to avoid making the discussions too abstract. Some examples of ontology mismatches discussed below are based on a comparison we conducted between two ontologies of university-student data. The two ontologies, one created at Liverpool University (henceforth Ont-L) and the other created at Cardiff University (henceforth Ont-C) were created for the same purpose and domain but were created independently of one another.

4.1 Conceptualisation mismatch

As stated in the previous section some conceptualisation mismatches are not easily recognised from the explicit ontology definitions. As a result, it is not easy to deal with such mismatches. However, we show below that some conceptualisation mismatches are easily recognisable in the explication process and can be dealt with rather easily.

Categorisation mismatch (*class mismatch*). The ontological mismatch between two conceptualisations that divide their entities into different subcategories may be difficult to resolve. Theoretically, the mismatch can be resolved by abstracting the knowledge of the two sets of subcategories to the abstraction level of the common class (see definition of categorisation mismatch). However, the common class could be a very abstract class (*e.g.,* the root of a class hierarchy), which implies that the abstraction process, and thus the mapping function, is a complicated and 'knowledge-intensive process'.

Aggregation-level mismatch (*class mismatch*). If two conceptualisations express information at different aggregation levels the mapping process requires the knowledge to be expressed at the same aggregation level. In general, this can be done either by abstracting the more detailed knowledge, or, by specialising the less detailed knowledge (or indeed by doing both).

Consider an example from Ont-C (the Cardiff ontology) and Ont-L (the Liverpool ontology). In Ont-C the term passed is used to denote that a student has passed a year of study. Ont-L distinguishes both the terms passed and deemed, the latter to denote that a student is deemed to have passed his year of study (*e.g.,* if a student has been sick and for that reason not able to meet all necessary formal requirements). Apparently, Ont-C uses a more abstract notion of 'having passed' than Ont-L. Roughly, we could say that passed (Ont-C) equals passed (Ont-L) plus deemed (Ont-L). A mapping from an expression in Ont-L onto an expression in Ont-C involves abstracting clauses passed (Ont-L) plus deemed (Ont-L) and mapping them onto the term passed (Ont-C) and is not likely to cause many difficulties (although information is lost as a result of the mapping, *cf.loss-less mappings*, [15]). However, the other way around, specialising the knowledge from Ont-C into Ont-L is more difficult because we do not know (in Ont-C) whether a student who passed his module should be considered as a passed (Ont-L) or as a deemed (Ont-L). Compare this to the discussion on attribute-type mismatches (see below).

Structure mismatch (*relation mismatch*). Providing mapping functions for structure mismatches in the general case is difficult. Having different relations between identical classes can be problematic but may not be in special cases. This depends on the case at hand. Consider, for instance, two ontologies both with classes warehouse and bricks. Suppose one ontology links these classes with the relation stores, and the other ontology links the classes with relation is-made-of. In this example, it is clear that the two relations should not be considered equal, and thus, they should not be mapped onto each other. The mismatch is not problematic because the two relations denote different ontological concepts, and so the two relations can coexist next to each other. Now, consider the house-and-brick example mentioned in section 3. Here, the two relations has-component and is-made-of denote different concepts with substantial overlap. Mapping from relation is-made-of onto relation has-component is possible, but the mapping in the other direction is not justified *per se*.

Attribute-assignment mismatch (*relation mismatch*). A mismatch in attribute assignment occurs when two conceptualisations differ in the way they assign an attribute class to other classes. Consider, a system s_1 that represents that X is a female student as *female_student(X)* and a system s_2 that represents this information as $student(X) \wedge female(X)$. In this case, the mapping between these two systems can be done by two simple mapping functions *mf* and *mf'* ($Ont(s_i)$ denotes the ontology of system s_i):

$$mf\,(Ont(s_1){:}\,female\_student(X)) \;\Rightarrow Ont(s_2){:}\,female(X) \wedge student(X)$$

$$mf'\,(Ont(s_2){:}\,female(X) \wedge student(X)) \;\Rightarrow Ont(s_1){:}\,female\_student(X)$$

A second example, taken from Ont-C and Ont-L, concerns the representation of the duration of a student's registration for a module (a module is a set of university courses). In Ont-C the duration of a registration is - implicitly - derivable from the attributes start-date and end-date linked to the class module. In Ont-L the duration is stored as a unit of time in a separate attribute duration linked to the class registration. Mutual translations between the two representations is not possible without information loss because the duration attribute does not state what the start and end dates are.

Attribute-type mismatch (*relation mismatch*). If two systems s_1 and s_2 both use term *mark(Module,Value)* but their value parameter is of a different type then a mapping is required between the value parameters. The mapping functions between the two systems are rather straightforward if both systems store exactly the same information in different formats. For instance, if system s_1 stores values as {A, B, C, D, E, F} and system s_2 stores values as {'A', 'B', 'C', 'D', 'E', 'F'}, then the mapping would require a function that maps A onto 'A', B onto 'B' etc., and a function that maps 'A' onto A etc. We note that a mapping function can involve some calculations. For instance, when two systems communicate, one measuring distances in miles, and the other measuring distances in kilometres the mapping function obviously requires some basic mathematical operators.

If both systems store the same information content then the mapping functions (both ways) will be rather straightforward. However, it is likely that two systems do not store precisely the same information content. For instance, if system s_1 stores percentages (*e.g.*, 75%) and system s_2 stores ratios (*e.g.*, 15/20) then there is a mapping function from system s_2 to system s_1, but not *vice versa*. The mapping function in such cases is one-directional.

4.2 Explication mismatch

An explication mismatch occurs when two ontologies have different definitions but their terms, their definiens, or their ontological concepts are the same. Below, we discuss how the six types of explication mismatches can be dealt with.

CT mismatch. If two systems use the same definiens *D* to denote two different concepts *C* and *C*' and refer to them with different terms *T* and *T*', respectively, then this mismatch probably does not require a mapping since the terms should be considered (and are already) different. However, the fact that both systems use the same definiens may indicate that there is another type of mismatch between the ontologies regarding the terms used in their definiens, for example (see also the discussion in section 5):

```
pub(X) :- bar(X)     ingot(X) :- bar(X)
```

CD mismatch. If two systems use the same term *T* while using different concepts *C* and *C*' and different definiens *D* and *D*' then it is clear that the expressions *T* should not be considered equal. To resolve such a mismatch the terms need to be renamed (at least one of them). For instance, Ont-C and Ont-L both use the term module but define their concepts differently. We could rename the terms to Ont-L-module and Ont-C-module, respectively (*cf.* [18]).

C mismatch. If two systems use identical terms *T* and definiens *D* but differ in the concepts *C*, then a mapping is required because the terms should be kept distinct in a cooperative context. The mismatch can be resolved by renaming the expressions (as in the CD mismatch).

TD mismatch. If two systems represent the same concept *C* but define it with different terms *T* and *T*' and definiens *D* and *D*' there are two possibilities. On the one hand, it can be argued that the terms refer to the same concept *C* for which reason the terms *T* and *T*' are considered synonymous and can be mapped onto each other. On the other hand, it can be argued that because the definiens *D* and *D*' are different the concepts denoted *C* are, in fact, not the same. Hence, there should be no mapping of the terms. Which solution more appropriate cannot be stated in the general case. This depends, among others, on the relative importance of the differences between *D* and *D*'. Consider the following two ontology definitions (in PROLOG format), defining the ontological concept 'information of a student comprises a unique identification, a name, and an address':

```
student_inf(I,N,A) :-   identification(I),  name(N),  home_address(A).
student_data(I,N,A) :-  id(I),  name(N),  residential_address(A).
```

These definitions have a TD mismatch. One can argue that (because the two definiens appear to define the same student information) the defined terms should be considered the same and can be mapped onto each other. However, one can also argue that a residential address is not necessarily the same as a home address, which makes the actually defined concepts different (despite the fact that both ontologies define the same ontological concept *C*). Hence, the terms should not be mapped onto each other.

T mismatch. If two systems use the same definiens *D* to denote a concept *C* but refer to it with different terms *T* and *T'* then the terms have to be mapped onto each other (*T* and *T'* are synonyms). In general, the mapping functions will be bi-directional. For instance, Ont-C refers to the university modules with the term number(X) (of type string), and Ont-L refers to this concept with the term code(X) (also of type string). Note, that their definiens are the same as they are both defined as strings. An example of two mapping functions (both ways) is:

mf (*Ont-C*: *number(X)*) ⇒ *Ont-L*: *code(X)*   *mf'*(*Ont-L*: *code(X)*) ⇒ *Ont-C*: *number(X)*

A potential problem occurs if the terms differ in the variables they distinguish. For instance, if one system uses the term student(Id, Name, Address) and another uses the term student(Id). Mapping the latter term onto the former term may cause a problem because Name and Address cannot be derived from the latter term.

D mismatch. It two systems define the same concept *C* and use the same term *T* but differ in their definitions then it can be argued that the concepts are in fact not the same. Hence, in a cooperative context, the terms should be kept distinct. A solution would be to rename the terms so that different terms are used. Alternatively, it can be argued that because both systems are intended to define the same concept *C*, the terms and definiens should be considered the same, which implies that two straightforward (identity) mapping functions (*i.e.,* one that links the same terms in different systems) would suffice. This would be the case with the following two definitions of humans (after Aristotle and Bergson, respectively):

```
human(X)  :-  featherless(X),  biped(X)          human(X)  :-  animal(X),  laughs(X)
```

5. Discussion

In this section we discuss four issues that arose as a result of an experiment performed with the Liverpool and Cardiff ontologies.

A first issue concerns the *inheritance of mismatches*. Consider the following situation in which we have ontologies A and B (assume FA(X) and FB(X) to be two formula, for which FA(X) ≠ FB(X)):

```
A-1:  a(X)  :-  b(X)          B-1:  a(X)  :-  b(X)
A-2:  b(X)  :-  c(X)          B-2:  b(X)  :-  c(X)
A-3:  c(X)  :-  FA(X)         B-3:  c(X)  :-  FB(X)
```

Both ontologies have three definitions, the first two of which are identical, but the last ones (*i.e.* definitions A-3 and B-3) have an explication mismatch. This mismatch is either of type D or of type CD (as we do not yet know whether the intended concepts are the same). Because there is no mismatch between definitions A-1 and B-1 we would - at first glance - say that a(X) in ontology A denotes the same concept as a(X) in ontology B. However, since a(X) is defined in terms of b(X), and b(X) is defined in terms of c(X) we find that a(X) refers to different extensions (concepts?) in ontology A and B. We refer to this as inheritance of mismatches. There are two possibilities now. As a first possibility, consider the following instantiations of the ontologies given above:

A-1:	animal(X) :-	bird(X)	B-1:	animal(X) :-	bird(X)
A-2:	bird(X) :-	cormorant(X)	B-2:	bird(X) :-	cormorant(X)
A-3:	cormorant(X) :-	eats_fish(X), flying(X)	B-3:	cormorant(X) :-	fish_eating(X), flies(X), dives(X)

For these ontologies, we may assume that the categories of animals in A and in B denote the same concept (there is a D mismatch between A-3 and B-3). Hence, the mapping between animal(X) in system A and animal(X) in system B is straightforward. In contrast to this instantiation, another possibility is that we have the following two ontologies, a heraldic and a biological instantiation of the two ontologies mentioned above:

A-1:	animal(X) :-	bird(X)	B-1:	animal(X) :-	bird(X)
A-2:	bird(X) :-	cormorant(X)	B-2:	bird(X) :-	cormorant(X)
A-3:	cormorant(X) :-	liver_bird(X)	B-3:	cormorant(X) :-	fish_eating(X), flies(X), dives(X)

These ontologies are different in that the denoted set of entities that satisfies predicate animal(X) are likely to be different in ontology A and ontology B (as liver-birds - bronze sculptures from which there are only two in the world - cannot fly, eat or dive there is a CD mismatch between A-3 and B-3). Whereas in the first instantiation of the ontologies, the concepts defined in A-3 and B-3 are the same (which makes it a D mismatch), in the second instantiation the concepts defined are not the same (which makes it a CD mismatch). Thus, dependent on the type of mismatch between A-3 and B-3, A-1 and B-1 have a C mismatch or do not have a mismatch at all. We conclude that the syntax of the ontology does not provide enough information to tell whether an inherited mismatch is a problematic (*i.e.* one that is easily resolved with a mapping function) mismatch or not.

A second - and related - issue concerns the *relations between different mismatch types*. Consider two ontologies which both have a definition of a class, referred to with the term *class*. The two classes have the following attributes (and attribute types), for instance:

Ontology A: *class*	Ontology B: *class*
attribute-1: String	*attribute-1: String*
attribute-2: Integer	*attribute-2: Integer*
attribute-3: String	*attribute-3: String*
attribute-4: String	*attribute-5: String*

Assume that the concept (*C*) that is defined here is the same in both ontologies. The definiens of the two classes are identical except for their last attributes. Thus, there is a D mismatch between the classes because both terms are equal (*viz. class*) yet *attribute-4* and *attribute-5* are not the same. Assuming these attributes to denote the same concept the mismatch between these attributes is a T mismatch. Note, that the D mismatch between the two classes is caused by a T mismatch between their attributes. Also, we may conclude that the mismatch is relatively easy to solve, namely by mapping the contents of the attributes that cause the mismatch. However, if there is no correspondence in the concepts denoted by these attributes, for instance, if *attribute-4* denotes the concept ‘previous address’ and *attribute-5* denotes ‘current address’ then there is a CT mismatch between the attributes[6]. Also, there is no obvious mapping between the two attributes (and thus, between the two classes). In conclusion, we remark that a D mismatch is easily solved if it is caused by a T mismatch, but not if it is caused by a CT mismatch (compare this with our findings in the first issue).

A third issue concerns our *notion of definitions*. We defined mismatch types assuming that an ontology contains definitions, which consisted of a term, a definiens, and a concept. We loosely defined the mismatches by saying that the definitions ‘differ in their term’ (or concept, or definiens). What exactly is to be considered different in this respect has not been defined. Consider the two following definitions of the term r(X):

```
r(X) :- q(X)          r(Y) :- q(Y)
```

Intuitively, one would say that these two definitions do not differ in their term and definiens (although one could argue that they are different because they are not fully identical). we assume the terms (and indeed the definiens and hence the definition) to be the same since variable names are not considered to be relevant in identifying terms or definiens. However, we have to be careful in applying this as a guideline. Consider the following definitions:

```
r(X:Integer) :-   q(X:Integer)          r(Y:Real) :- q(Y:Real)
```

In this case we do have a relevant mismatch. There is an attribute-type mismatch and hence, the definitions should not be considered identical. Note that we would not even have to use a typed logic to illustrate this problem, simply *assuming* the types to be different in different systems would suffice. Differences in the possible instantiations of X and Y could also be determined by differences in the definitions of q(X). In that case, we would have an inherited mismatch. In conclusion, we state that we can only detect a mismatch between two expressions given the context in which the expressions are used, again we note that syntax does not suffice. Also, whether two expressions cover the same concepts (and thus, whether they can be mapped onto each other) depends in the end, on the textual descriptions of all terms that directly or indirectly contribute to the meaning of the expressions (*cf.* inherited mismatches).

A final - and related - issue also concerns our notion of definitions. In particular, we note that our *explication mismatch types are defined on form, rather than on content*. This means that two definitions expressed in one language could have different mismatch types if they were expressed in another language. Consider the following two sets of definitions, the first expressed in P/FDM ([3]), and the second expressed in PROLOG:

```
content(module) ->        string          content(X,Y) :- module(X), string(Y)
content(module_desc) ->   string          content(X,Y) :- module_desc(X), string(Y)
```

[6] One could argue that in that case, there is a CD mismatch between the entities and not a D mismatch. This is not necessarily true. For instance, if *attribute-4* denotes the date of birth of a person, and *attribute-5* denotes the age of a person, then one could say that the attributes denote different concepts, but the (more general) classes denote the same concept. Note, that this depends on how the concept is described (*e.g.,* in natural language).

The two definitions in P/FDM format have a mismatch in terms but not in their definiens, whereas the two definition in PROLOG format have a mismatch in the definiens, but not in their terms. Thus, by translating a P/FDM expression into a PROLOG expression, a T mismatch becomes a D mismatch. Although we illustrate this issue with two different languages, even within one language there are several possibilities to explicate a conceptualisation. This means that even within a language we can, dependent on the way we explicate, get different (explication) mismatches.

6. Conclusions

The idea behind the study discussed here is to identify a set of heuristics that allow us to determine whether systems can join a cooperative community, or, to provide guidance for the design of such systems. In this section we present some tentative conclusions with respect to the difficulty of dealing with ontological mismatches. We list the ontological mismatches of our classification and divide them into three categories: (*a*) *manageable*, (*b*) *hard*, and (*c*) *context dependent* (the latter meaning that the difficulty depends on the case at hand). The list reflects a first impression of the authors on the solvability of the mismatches. It is therefore tentative.

(*a*) *manageable:* T, CT, CD, C, and attribute-type mismatch
(*b*) *hard*: categorisation mismatch, structure mismatch
(*c*) *context dependent*: TD, D, attribute-assignment and aggregation-level mismatch

Not all mismatches are equally likely to occur. For instance, a CT mismatch is not likely to occur, but a categorisation mismatch is very likely to occur. Also, the mismatches differ in the ease with which one can detect them. For instance, a C mismatch is probably difficult to detect, whereas T and CT mismatches are easy to detect. Finally, we recap five conclusions that we derive from our analysis.

- The syntax of the knowledge base does not provide enough information to tell whether an inherited mismatch is an easily mappable mismatch.
- A D mismatch is easily solved if it is caused by a T mismatch, but hard to solve if it is caused by a CT mismatch.
- Certain mismatches between two expressions can only be detected from the context in which the expressions are used, as the syntax does not suffice to detect all mismatches.
- Whether two expressions cover the same concepts (and thus, whether they can be mapped onto each other) depends in the end on the natural language descriptions of all terms that directly or indirectly contribute to the meaning of the expressions.
- Translating ontologies which have a T mismatch from one language into another language, may cause the resulting knowledge bases to have a D mismatch.

Acknowledgements

The KRAFT project is funded by BT and the Engineering and Physical Sciences Research Council in the UK (EPSRC). Besides these organisations, the authors wish to thank Martin Beer, Ken Chan, Zhan Cui, Bernard Diaz and Ken Lunn for their contribution to this article.

References

[1] Batini, C., M. Lenzerine, and S.B. Navathe (1986). Comparison of Methodologies for Database Schema Integration, *ACM Computing Surveys*, Vol. 18., No. 4, pp.323-364.

[2] Ceri, S., and J. Widom (1993). Managing Semantic Heterogeneity with Production Rules and Persistent Queues, *Proceedings of the 19th VLDB Conference*, Dublin, Ireland, pp.108-119.

[3] Embury, S.M. (1995). *User Manual for P/FDM V9.0*, Technical Report AUCS/TR9501, Dept. of Computing Science, University of Aberdeen, Aberdeen, Scotland.

[4] Farquhar, A., R. Fikes, and J. Rice (1997). The Ontolingua Server: A Tool for Collaborative Ontology Construction, Int. Journal of Human-Computer Studies, 46, pp.707-727.

[5] Gray, P.M.D., A. Preece, N.J. Fiddian, W.A. Gray, T.J.M. Bench-Capon, M.J.R. Shave, N. Azarmi, M.Wiegand, M. Ashwell, M. Beer, Z. Cui, B. Diaz, S.M. Embury, K. Hui, A.C. Jones, D.M. Jones, G.J.L. Kemp, E.W. Lawson, K. Lunn, P. Marti, J. Shao, and P.R.S. Visser (1997). *KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases*, DEXA '97, Toulouse, France.

[6] Gruber, T.R. (1992). *Ontolingua: A Mechanism to Support Portable Ontologies, Version 3.0*, Knowledge Systems Laboratory, Stanford University, Stanford, California, United States.

[7] Gruber, T.R. (1995). Toward principles for the Design of Ontologies Used for Knowledge Sharing, *Int. Journal of Human-Computer Studies*, Vol.43, pp.907-928.

[8] Guarino, N. (1994). The Ontological Level, *Philosophy and the Cognitive Sciences*, Vienna, Hölder-Pichler-Tempsky.

[9] Guarino, N. (1995). Formal Ontology, Conceptual Analysis and Knowledge Representation, *International Journal of Human-Computer Studies*, International Journal of Human and Computer Studies, special issue on *The Role of Formal Ontology in Information Technology*, N. Guarino and R. Poli (*eds.*), Vol 43, No. 5/6.

[10] Kitakami, H., Y. Mori, and M. Arikawa (1996). An Intelligent System for Integrating Autonomous Nomenclature Databases in Semantic Heterogeneity, *Database and Expert System Applications, DEXA'96*, Zürich, Switzerland, pp.187-196.

[11] Kim, W. and J. Seo (1991). Classifying Schematic and Data Heterogeneity in Multidatabase Systems, *IEEE Computer*, Vol. 24, December 1991, pp.12-18.

[12] Levesque, H.J., and R.J. Brachman (1985). A Fundamental Tradeoff in Knowledge Representation and Reasoning (Revised Version), *Readings in Knowledge Representation*, R.J. Brachman, and H.J. Levesque (eds.), pp.42-70, Morgan Kaufmann, San Mateo, California, United States.

[13] March, S.T. (1990). *Special Issue on Heterogeneous Databases*, ACM Computing Surveys, ACM Press, Vol. 22, No. 3.

[14] Renner, S.A., A.S. Rosenthal, and J.G. Scarano (1996). Data Interoperability: Standardization or Mediation, The MITRE organisation: http://www.nml.org/resources/misc/-metadata/proceedings/metadata/renner/.

[15] Sciore, E., M Siegel, and A. Rosenthal (1994). Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems, *ACM Transactions on Database Systems*, Vol.19, No.2, pp.254-290.

[16] Shaw, M., and B. Gaines (1989). Comparing conceptual structures: consensus, conflict, correspondence and contrast, *Knowledge Acquisition,* 1, 341-363.

[17] Visser, P.R.S., D.M. Jones, T.J.M. Bench-Capon and M.J.R. Shave (1997). An Analysis of Ontology Mismatches; Heterogeneity versus Interoperability, *KRAFT Report KP6*, also appeared in *AAAI 1997 Spring Symposium on Ontological Engineering*, Stanford University, California, USA, pp.164-172.

[18] Wiederhold, G. (1994). Interoperation, Mediation, and Ontologies, *Proceedings International Symposium on Fifth Generation Computer Systems* (FGCS94), Workshop on Heterogeneous Cooperative Knowledge Bases, Vol. W3, pp.33-48, ICOT, Tokyo, Japan.

[19] Woods, W.A. (1985). What's in a Link: Foundations for Semantic Networks, *Readings in Knowledge Representation*, R.J. Brachman, and H.J. Levesque (*eds.*), pp.217-241, Morgan Kaufmann, San Mateo, California, United States.