On Mismatches Between Ontologies
A first investigation into knowledge fusion

Pepijn R.S. Visser and *Dean M. Jones*

KRAFT Working Paper KP100, v2.3 - PRSV 2.3 / Oct96
Department of Computer Science
University of Liverpool
P.O. Box 147, Liverpool L69 7ZF
{pepijn, dean}@csc.liv.ac.uk

On Mismatches Between Ontologies
A first investigation into knowledge fusion

Pepijn R.S. Visser and *Dean M. Jones*

KRAFT Working Paper KP100 v2.3 - PRSV 2.3 / Oct96
Department of Computer Science
University of Liverpool
P.O. Box 147, Liverpool L69 7ZF
{pepijn, dean}@csc.liv.ac.uk

Abstract

In this paper we discuss mismatches between two database schemes (here referred to as ontologies), produced for the UNIKb prototype. We present a (new) classification of system-communication mismatches after we discuss the occurrence of these mismatches between the Cardiff University and Liverpool University ontologies. Thereafter, we investigate to what extent the mismatches can be resolved. We define terminology to be able to give a more accurate description of ontology relations. Using this terminology, we draw some preliminary conclusions regarding ontology mappings. In four appendices we present the Cardiff ER diagram and ontology (expressed in P/FDM), and the Liverpool ER diagram and ontology (expressed in P/FDM).

1. Introduction

The KRAFT project aims at creating an architecture for multi-site (heterogeneous) information systems. This requires knowledge that is located at different sites to be fused. The project has started in May 1996 and has produced some initial ideas on the construction of the architecture of such information systems (Embury *et al.*, 1996). To test these ideas it has been decided to create a prototype system. The prototype, named UNIKb (after UNIversity Knowledge base), consists of three cooperating databases each located at a different university site. All three databases contain data on university students, such as their registration details and their progress reports. The aim is to use one common database schema for all three sites. Each site will then represent its local university regulations by defining their own constraints over the common schema (in the remainder we refer to a database schemas with the more general term ontology).

Communication between two heterogeneous systems is impeded by, amongst other things, the problem that a term used in one system may have a different meaning in another system. The term cannot be used in system communication because it is ambiguous. One way to deal with this problem is to rename all terms so as to make them unique (*e.g.,* by giving them a system-related prefix). This however is only a partial solution and does not allow adequate system communication. The problem is that the renamed terms do not make sense to the other system. If we require a system to *use* information from another system we have to express the knowledge from the supplying system in terms of the knowledge of the receiving system. Otherwise stated, the knowledge from the supplying system has to be *fused* with the knowledge from the receiving system.

This paper reports on a first investigation into knowledge fusion. Although we discuss the ontology of the UNIKb prototype, our primary focus is the knowledge exchange between two systems with *different* ontologies (we note that since the UNIKb prototype assumes one single ontology it is less suitable for our current aim). Our point of departure is the two different university ontologies that were produced in an earlier phase for the prototype at Cardiff and Liverpool University, respectively. In this paper we discuss these ontologies, investigate their ontological mismatches and the possibilities to resolve these mismatches. In the long term (not in this article) we aim at finding guidelines for ontology mappings. That is, (heuristic) rules that describe the conditions under which mappings between different ontologies are possible. These rules could then be used to define the requirements for systems to participate in a cooperative environment, such as the KRAFT network.

This paper is structured as follows. In section 2 we discuss the types of communication mismatches that can occur between two systems. In particular, we extend and classify the list of mismatches distinguished in postition paper KP3 (Visser, 1996). In section 3 we briefly compare the two ontologies as produced at Cardiff and Liverpool University (the schemas and ontologies are given in full in the appendices). Then, in section 4 we analyse the two ontologies with respect to the mismatch types of section 2. After this, in section 5, we discuss the possibilities to overcome these mismatches. In section 6 we define terminology which facilitates describing mapping relations between ontologies. Finally, in section 7 we discuss seven issues that arise from the previous sections, we rank the mismatch types according to their solvability, and we draw ten tentative conclusions. In the appendices we provide the ER diagram of the Cardiff ontology (appendix A), a P/FDM specification of the Cardiff ontology (appendix B), the ER diagram of the Liverpool ontology (appendix C), and a P/FDM specification of the Liverpool ontology (appendix D).

2. A classification of communication mismatches

Communication between heterogeneous systems is, amongst others, impeded by mismatches in their underlying assumptions. In KP3 (Visser, 1996) a set of communication mismatch examples was listed. In this section we present a revised version of this list. In particular, we extended the list of mismatch examples, after which we structured them. The classification will be used in the remainder of this paper to analyse the mismatches between the two ontologies. As with the previous list of mismatch examples, we do not claim the classification presented below to be complete nor disjoint (*cf.* Visser, 1996). However, we deem the classification to be a useful instrument to determine what mismatches are present between ontologies, to determine what mismatches are hard to resolve, and, to define guidelines with which ontologies can be created.

The basic categories of communication mismatches between two systems are the (1) *ontology mismatch*[1], and the (2) *language mismatch*.

(*1*) *ONTOLOGY MISMATCH*

An ontology mismatch occurs when two systems have modelled their knowledge using different ontologies. In particular, the knowledge bases differ in their underlying (*1.1*) entities, (*1.2*) relations, or (*1.3*) in the way they define their knowledge.

(*1.1*) *Entity mismatch*

An entity mismatch occurs when two systems differ in the way they conceptualise the entities in the world.

(*1.1.1*) *Categorisation mismatch*

A categorisation mismatch occurs when two systems contain knowledge about the same entity but split this entity into different subcategories[2]. The subcategory division can be made using relations such as, *component-of*, *part-of*, *is-a*, or *instance-of*. As an example we consider two knowledge bases that both contain knowledge about animals. A categorisation mismatch occurs when one knowledge base structures its knowledge around entities mammals and birds, whereas the second structures its knowledge around entities carnivores and ungulates.

---

[1] We assume the conceptualisations of the systems to be explicit , *i.e.* we assume an ontology of the system to exist.

[2] See also the discussion in section 7.

(*1.1.2*) *Aggregation-level mismatch*

An aggregation-level mismatch occurs if two systems both contain knowledge on the same entity, but at different levels of abstraction. For instance, one knowledge base that contains knowledge about the male population, say: males(X) ← obliged_to_join_the_civil_service(X) and a database that distinguishes between young and old males, say: young_males(3.200.000), and old_males(12.000.000).

(*1.2*) *Relation mismatch*

A relation mismatch occurs when two systems differ in the way they conceptualise the relations between the entities in the world.

(*1.2.1*) *Attribute-assignment mismatch*

An attribute-assignment mismatch occurs when two systems both contain knowledge about a concept but differ in the way they assign their attributes to the concept. An example is the concept of a (red) car. One knowledge base could represent a red car as red_car(X), whereas another knowledge base could represent it as car(X) ∧ red(X). We note that the two conceptualisations not necessarily have a categorisation mismatch.

(*1.2.2*) *Attribute-type mismatch*

An attribute-type mismatch occurs when two systems use the same terms (predicate relation) but differ in the type of the attributes assigned. Consider the following two typed expressions: length(X:Road, L:Kilometres), and length(X:Road, L:Miles).

(*1.3*) *Definition mismatch*

A definition mismatch occurs when two systems contain different definitions while using either the same definiendum, while using the same definiens, or while attempting to define the same concept in the real world. Here, we consider a definition to be composed of a *definiendum* (*i.e.* denoting an atomic formula in a formal language), a *definiens* (*i.e.* a (compound) formula in a formal language) and a *concept* in the world (which we - for practical reasons - deem to be expressed in natural language). We abbreviate the three elements of a definition with *C* (for concept), *T* (for definiendum - to avoid confusion with the definiens, we use the word 'term'), and *D* (for definiens). An example is the definition of the concept 'vessel' (*C*) as vessel(X) ← seagoing(X) ∧ large(X), in which vessel(X) is the definiendum (*T*), and seagoing(X) ∧ large(X) is the definiens (*D*).

(*1.3.1*) *CT mismatch* (or *Concept and Term mismatch*)

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

(*1.3.2*) *CD mismatch* (or *Concept and Definiens mismatch*)

A CD mismatch occurs when two systems use the same term *T* but differ in the concept *C* they are intended to define and the definiens *D* used for the definition. Consider the term mitre. One system may define the concept of the head of a bishop with the expression: mitre(X) ← head_dress_of_bishop(X), whereas a second system may define the concept of a straight angle joint of wood with the expression: mitre(X) ← straight_angle_joint_of_wood(X). Although both systems use the same term *T* it is clear that the systems denote a distinct concept and use a different definiens.

(*1.3.3*) *C mismatch* (or *Concept mismatch*)

A C mismatch occurs when both systems have the same term *T* and definiens *D* but differ in the concept they are intended to define. For instance, both systems could use the expression red_mitre(X) ← mitre(X) ∧ red(X) while one system is intended to define the concept of a red piece of wood whereas the other system is intended to define the concept of a red hat.

(*1.3.4*) *TD mismatch* (or *Term and Definiens mismatch*)
A TD mismatch occurs when two systems 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 systems 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 systems differ both in their term *T* and their definiens *D*.

(*1.3.5*) *T mismatch* (or *Term mismatch*)
A T mismatch occurs when two systems define the same concept *C* using the same definiens *D* but refer to it with different terms. This mismatch occurs when systems use synonyms. Consider the use of the expression: ship(X) ← seagoing(X) ∧ large(X) by one system and vessel(X) ← seagoing(X) ∧ large(X) by another system, both defining the concept of a ship.

(*1.3.6*) *D mismatch* (or *Definiens mismatch*)
A D mismatch occurs when two systems define the same concept *C* and use the same term *T* to refer to the concept but use a different definiens. For instance, one system may define the concept of term mitre with the expression: mitre(X) ← head_dress_of_bishop(X), whereas a second system may define the same term with the expression: mitre(X) ← hat_of_bishop(X) ∧ deeply_cleft(X).

(*2*) *LANGUAGE MISMATCH*
A language mismatch occurs when two systems use different languages to express their knowledge. This is a very broad category of communication mismatch problems. For instance, one system uses a production-rule system (*e.g., if* rain *then* wet_street) and another system uses a horn-clause knowledge system (*e.g.,* wet_street ← rain).

(*2.1*) *Mismatch in formalism expressiveness*
A mismatch in formalism expressiveness occurs when two systems use different formalisms to express their knowledge while the expressiveness of the languages differs. Take as an example a system using predicate logic (*e.g.,* ∀ X: flies(X) ← bird(X)) and another system that uses propositional logic (*e.g.,* flies ← bird) (note that this type of mismatch is an instance of the previous type of mismatch).

(*2.2*) *Mismatch in computational tractability*
A mismatch in computational tractability occurs when two systems use languages with a different computational tractability. Take as an example a system using predicate logic (*e.g.,* ∀ X: flies(X) ← bird(X)) and another system that uses production rules (*e.g., if* bird(X) *then* flies(X)).

The classification of mismatches, and in particular, the classification of language mismatches are still under revision. In the remainder of this article we focus on ontology mismatches.

3. A brief comparison of the Cardiff and Liverpool ontologies

Before we present an analysis of specific mismatches between the Liverpool and Cardiff ontologies (henceforth Ont-L and Ont-C), we will first discuss in a more general sense some of the differences between them, especially with respect to the motivations for their design and the consequences that result. By doing so, we hope to aid the reader in understanding how such mismatches can arise. Some discussion of Ont-L is given, but we do not analyse Ont-C in great depth. For more detailed information see the appendices and KP4 (Ashwell *et al.,* 1996).

The designs of the ontologies for the UNIKb prototype system can be assessed according to whether they satisfy two distinct criteria:

(i) adequately testing the KRAFT architecture, and
(ii) performance of the required tasks correctly.

Although there are other factors that would normally be used as criteria for assessing database systems (such as maximising efficiency) where these do not interfere with our primary aims, they can safely be ignored for present purposes. As we have not yet defined precisely what aspects of the KRAFT architecture the prototype is designed to test, we can use criterion (i) to assess the design only in the most general terms. We will concentrate mainly on criterion (ii), and return to criterion (i) at the end of the section.

In using criterion (ii) to assess the system, we first need to identify exactly what data we should be able to query the local systems for, as any representation should be assessed in relation to the task that it is to be used for (Visser 1995). As stated in KP4 (Ashwell *et al*., 1996), the proposed system would enable data relating to the academic record of a student at a particular institute to be maintained as well as allowing access to course information. In order that two ontologies can be compared, we feel that we need to be slightly more precise about defining tasks that the system will be able to perform. To this end, we have drawn up the following list:

1. *Assess Validity of Transfer*: determine whether it is possible for a student to transfer to a degree from a similar degree at some other university.
2. *Assess Submitted Registration*: determine whether the proposed registration of a resident student is valid.
3. *Assess Pre-university Qualifications*: determine whether a prospective student meets the entry requirements.
4. *Assess Intra-university Qualifications*: determine whether, given a set of module marks, a student can progress to a subsequent year of study.

3.1 Motivations for the ontology designs

The main reason that the ontologies exhibit various differences is that they were designed to store slightly different sets of data. It seems likely that Ont-C was designed with the stereotypical student in mind. This is probably more in keeping with criterion (i) above than criterion (ii) and, for the design of the prototype, this is a reasonable simplification. However, the need to analyse some of the problems in resolving mismatches between heterogeneous systems requires that another ontology be developed independently of Ont-C. The design for Ont-L was motivated from the opposite perspective to Ont-C[3], *i.e.* it should be able to store student records that do not necessarily conform to some typical example. For example, consider the case of a student who has failed a year and is invited to resit some modules. In representing such a case according to Ont-C, it is not immediately obvious whether the value of the relation year in the Student entity now refers to:

(a) the number of years that the student has been in attendance at a university (year of residence), or
(b) the part of the degree that is currently being studied for (year of study).

The difficulty is that each of these interpretations is problematic. Under interpretation (a), there is no record of the part of the degree course that is currently being taken, which is obviously necessary if tasks 1,2 and 4 are to be performed. Assuming therefore that this interpretation is incorrect, we can now analyse the problems associated with interpretation (b).

[3] It is important to make clear that this difference in perspective was only evident *after* the design of Ont-L and that at no stage was a decision made that the schema should necessarily be different. Indeed, the initial design of Ont-L was done without knowledge of the details of Ont-C

Given the need to solve tasks of type 4, it would be advantageous to keep a record of a student's progress in a year of study. The entity Module in Ont-C has an associated mark, making it possible to record the fact that a module has previously been taken along with the corresponding result that was achieved. However, the queries that would be necessary in order to be able to decide whether a particular year of study has been passed are more complex than would be the case if the results for a year of study were recorded together.

Also, in performing tasks 1 and 2, it will be necessary to record somewhere the combinations of modules that form an acceptable year of study. It seems that the initial expectation as outlined in KP4 (Ashwell *et al.*, 1996) was that this information would be expressed in the form of constraints on the data. As the course information relating to such combinations is quite complex, it is probable that such constraints will also need to be quite complex. In responding to queries of task type 1, this information needs to be transferred between sites and we anticipate that attempting to transfer overly complex constraints may initially prove to be quite difficult. For the purposes of the prototype (we now assess the design according to criterion (i) above) it may be a requirement to keep the transferred information as simple as is practicable.

We are not arguing here that one or other of the ontologies should be used in the prototype but rather highlighting the differences between them, and the consequences of implementing the designs in relation to criterion (i) above. These can be summarised in terms of a trade-off. On one hand, we have a simpler ontology, Ont-C, that would be fairly straightforward to implement at the different sites but that probably requires a set of complex constraints - especially in representing the course information - coupled with complex queries necessary to perform some of the required tasks. On the other hand, we have a slightly more complex ontology - Ont-L - that enables us to simplify the constraints on the data and which allows the required information to be uncovered with less difficulty.

3.2 The design of Ont-L

The differences that are evident between Ont-L and Ont-C can be summarised if we separate two aspects of the domain. Firstly, we have the course information that is essentially static, changing only under relatively infrequent conditions, such as the introduction of a new course. Secondly, we have the information relating to individual students which is more dynamic, resulting in more frequent updates. The data that relates to a student has a temporal aspect to it as it will change through the course of an academic session, whereas course information will remain the same. These sub-domains each correspond loosely to two of the tasks identified above. Tasks 1 and 2 are concerned largely with querying the course data, whereas performing tasks 3 and 4 mainly requires querying the student data.

In representing the course information, the first and most fundamental departure from Ont-C is the decision to include the entity Year_of_study_template. This entity is used to record all known combinations of courses that satisfy the various course rules such as necessary pre- and co-requisites. If a submitted registration conforms to a recognised template, then it can be accepted as a valid registration merely by querying a student's individual record to ensure that all relevant courses have been passed. If a submitted registration is not recognised as a pre-existing template, then the rules that govern the assessment of registrations can be invoked. If it found to be valid, Year_of_study_template can be updated with the new record. As each new template is generated, it is given a unique code that acts as a key for the entity.

For the recording of the student data, we have introduced the entity Year_of_study. As outlined above, this helps us to maintain the academic records of some kinds of non-standard student and alleviates somewhat the difficulties associated with tasks of type 4 above. We also include the entity Registration as the set of modules that a student takes during any one unit of time, whether that be a term, semester, academic year or some other duration. This further facilitates accounting for non-typical students, as we can now record information relating to resitting students where a partial registration may be necessary..

4. Mismatches between Ont-C and Ont-L

We now present some examples of communication mismatches between Ont-C and Ont-L, based on the classification that was detailed in Section 2. This is done with the aim of illuminating some of the difficulties involved in knowledge sharing and fusion, even when the communicating systems were designed to handle very similar data (subject to the differences outlined in section 3.1) and to perform the same system tasks. Note that, in the context of the prototype, we are not concerned with language mismatches as the local sites will be using the same representational formalism (P/FDM). Therefore, all of the mismatches given here are ontology mismatches.

4.1 Entity mismatches

In looking for this kind of mismatch, we are trying to identify differences in the basic classifications that are made in a domain. As outlined in section 3.1, Ont-C and Ont-L were designed for slightly different student models. It might be expected that there would at least be agreement on the basic entities that exist in the domain, and indeed there is much that is common between the ontologies. However, Ont-L was designed with a slightly broader view of the data to be handled, and this is often reflected in a description of the domain that is more generally applicable.

4.1.1 Categorisation mismatches

An example of this mismatch type can be seen if we consider the entire ontologies as the top level objects in a hierarchy (in this case a part_of relationship exists between the objects in adjacent levels). In Ont-C we find that the second level can be seen to contain the entities Student, Module, Module_desc and Mark[4]. In Ont-L we can identify Student, Year_of_study, Year_of_study_template, Registration, Module, Mark and Student_mark as being a corresponding decomposition. It is clear, however, that as the entities that are identified as being part of the domain differ in Ont-C and Ont-L, a categorisation mismatch exists between them.

4.1.2 Aggregation-level mismatches

In Ont-C, whether a student has passed a module or not is easily recorded by including the module in the passed relation. As Ont-L strives for more generality, recording the same information is not quite as simple. Here, there are two relevant relations, passed and deemed. This reflects the situation that a student whose numerical score in a year of study may not be sufficient to automatically pass that year of study, can still be deemed to have passed the year through additional assessment, such as a *viva voce*.

As noted in section 3.1, one of the motivations for the design of Ont-L was the confusion surrounding the meaning of the Year relation in Ont-C. This led to the decision to introduce a new concept for each of the two possible interpretations a Year_of_study and a Year_of_residence. For the typical student that will be represented in Ont-C, no such distinction is necessary as the year of study and the year of residence will refer to the same period of time. Here we have another aggregation level mismatch as the concept of a year has been represented at a further level of detail in Ont-L.

As an example of how mismatch types do not have definite boundaries, consider how time is represented in the ontologies. Ont-C records the start_date and end_date of a module taken by a student, whereas Ont-L records that a registration lasts for a certain length of time, whether that be a year, a semester or whatever. It could be said that this is a categorisation mismatch, in that there are two basic "time entities", a date and a duration. However, another interpretation could be that, in Ont-C, dates always come in pairs and as such are used to represent a duration, with a further level of detail added. Under this interpretation, we have an aggregation-level mismatch

4 We use the phrase "can be seen to contain" advisedly as, in the absence of an explicit definition of such a hierarchy, it could be constructed in more than one way.

4.2 Relation mismatches

We now give examples of the mismatches that can arise from the different ways in which attributes can be represented.

4.2.1 Attribute-assignment mismatches

In talking about the different ways in which time is dealt with in section 4.1.2, we did not mention that, in Ont-C, a length of time is associated with a module whereas in Ont-L it is a registration, rather than each module, that lasts for some period of time. The duration of a module in Ont-L is implicit in the duration of the registration in which the module is taken. Thus, the duration of a module is associated with different entities and we have an attribute-assignment mismatch.

Another example of this mismatch type is closely related to the categorisation mismatch highlighted in 4.1.1. In Ont-L there is no corresponding (explicit) relationship between Student and Module to the one that is expressed by passed in Ont-C. This relationship is defined in Ont-L in the value of pass_status defined on the entity Student_mark. Again we see that a relation that expresses the same information is defined in two distinct ways in Ont-L and Ont-C.

4.2.2 Attribute-type mismatches

The possibility of this type of mismatch arising was a motivating factor for the definition of the value entity Mark in Ont-L. Consider the various ways in which a mark can be assigned to some piece of work:

- as a percentage,
- as a score based on some other ratio (*e.g.* 15/20),
- based on the class system (I, IIi, . . .),
- as an alphabetic grade (A, B, . . .).

There are two interrelated problems here. Firstly, although a percentage and a score out of twenty are both numerical values, they are based on different ratios. The second problem is that it may sometimes be necessary to translate a numerical value into an alphabetic grade or class. In designing Ont-L, it was decided that a mark could be given as a percentage and/or a grade (as long as the threshold attribute of Module is given in both formats, the pass_status of Student_mark can be determined from either). In the case where only a grade is given, we have an attribute-type mismatch with Ont-C, where a mark is defined as an integer, referring to a percentage.

4.3 Definitional mismatches

The classification we gave in section 2 simplifies the process of identifying and categorising (and hopefully resolving) certain kinds of mismatches, as it reduces the problem to obtaining answers to three questions:

1. Is the concept different?
2. Is the term different?
3. Is the definiens different?

4.3.1 CT mismatches

What is considered to be a CT mismatch depends on our position regarding the interpretation of a definiens. If we held *type definitions* of an entity attribute to be part of the definiens, then there are many examples we could include as CT mismatches (*i.e.* all entity attributes with the same type). If we did not consider a type definition as a part of the definiens, it is unlikely that we would find any examples of CT mismatches. Here, we take the former view and choose to include the type of an attribute to be part of the definiens. This decision is justified as a definition of type limits the range of values of an attribute. Note, that this is consistent with the inclusion of the syntactic constraints in the P/FDM scheme (see Appendix D).

4.3.2 CD mismatches

It commonly occurs that the same term is used to refer to different concepts. In these circumstances, the corresponding definition will usually also be different. Consider the entities that are labelled Module in the ontologies. In Ont-C, this entity is used to record the modules that an individual student has taken or has passed, whereas in Ont-L, it corresponds to the description of a module independent of any particular student. This is a mismatch in the concept and the definiens. This example highlights the fact that, within a domain such as that of the UNIKb prototype, it is possible that the same term is used to denote different, but not disjoint, concepts. In such cases, it may not be immediately obvious that different concepts are being represented. For example, consider the two definitions of Mark. In Ont-C, this denotes the mark that a student has attained at some module, whereas in Ont-L, it is a generic type that enables us to record marks in terms of percentages and also in terms of alphabetic grades. This is also a CD mismatch as the entities are intended to represent different concepts.

4.3.3 C mismatches

In the Ont-C definition of the term Module there is an attribute name that is used to record the names of modules that have been passed or are currently being taken by a student. The Ont-L definition of Module also has an attribute name which is used to record the names of all available modules. The term that is used to refer to this attribute is the same in each ontology (*e.g.* in P/FDM format, it is name(Module)) and both are defined to be of type string. We therefore have a mismatch in the concept.

4.3.4 TD mismatches

As noted in section 4.3.1, the term Module is used to refer to different concepts in the two ontologies. The concept that is referred to by Module in Ont-L corresponds to the concept that is referred to by the term Module_desc in Ont-C, *i.e.* the description of a module. The definiens for Module and Module_desc are different, as modules are formally described in different ways at the two sites and so we have an example of a TD mismatch.

A similar relationship exists between Student_mark in Ont-L and Mark in Ont-C. Both entities are used to record the mark for a particular student at a particular module. Therefore we have the same concept being represented with different terms and with different definiens *i.e.*, a TD mismatch. A discussion of the relationship between CD and TD mismatches is given in section 7.

4.3.5 T mismatches

University modules can usually be referred to with a unique alpha-numeric code. Both ontologies attempt to model this information in the module description entities, Ont-C with number and Ont-L with code. As both of these attributes are defined as type string, we have a mismatch in terms but not in concept or definiens *i.e.* a T mismatch. A similar example can be seen in the attributes termed content in Ont-C and description in Ont-L, again defined on the module description entities.

4.3.6 D mismatches

A simple example of this mismatch type occurs between the entities Degree. Here we have slightly different definitions of the same term which attempt to represent the same concept (the set of available degrees). Another example can be seen by comparing the definitions of the terms Student. In both Ont-L and Ont-C, this entity is used to record the details of individual students. The ontologies differ however on the details that are to be included in these records and again we have a D type mismatch.

5. Resolving ontological mismatches

In the previous section we described several examples of mismatches between Ont-C and Ont-L. The purpose of this section is to investigate to what extent these mismatches can be resolved. For each type of mismatch discussed in section 2 we discuss whether they can be resolved, and if so, how we could define a suitable mapping function. An additional purpose of this section is a preliminary investigation into the requirements for adequate ontology mappings. The structure of this section resembles the ontology-mismatch classification presented in section 2.

5.1 Entity mismatch

If two systems differ in the way they conceptualise the entities in the world they are said to have an entity mismatch. In section 2 we distinguished between categorisation mismatches and aggregation-level mismatches.

Categorisation mismatch

The ontological mismatch between two systems 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 entity (see definition of categorisation mismatch). However, the common entity could be a very abstract (the most abstract) entity, which implies that the abstraction process is a complicated and 'knowledge-intensive process'. Probably, the translation from an ontology into an ontology with another entity subdivision - if possible - has to be done for each pair of (instantiated) expressions of the respective knowledge bases.

If we reconsider the entity mismatch in the UNIKb example (section 4.1) we find that in Ont-C, the most abstract entity, say UNIKb_ knowledge, contains sub entities Student, Module, Module_desc, and Mark, whereas in Ont-L UNIKb_knowledge contains sub entities Student, Year_Of_Study, Year_Of_Study_Template, Registration, Module, Mark, and Student_Mark. If we would abstract both set of sub entities to a common entity we arrive at the most abstract entity UNIKb_knowledge. Arguably, the definition of the abstraction function here is indeed a very complex and knowledge-intensive process.

Aggregation-level mismatch

If two systems represent information at different aggregation levels the mapping process requires the knowledge to be expressed (expressible) at the same aggregation level. In general, this can be done either by abstracting the most detailed knowledge, or, by specialising the least detailed knowledge (or indeed by doing both). Consider an example from the UNIKb Ont-C and Ont-L. In Ont-C the term passed is used whereas in Ont-L both the terms passed and deemed are used. Ont-C has a more abstract notion of passed than Ont-L. Roughly, we could say that passed (Ont-C) equals passed (Ont-L) plus deemed (Ont-L). A translation from Ont-L into 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. 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 mismatch.

5.2 Relation mismatch

If two systems differ in the way they conceptualise the relations in the world they are said to have a relation mismatch. In section 2 we distinguished between attribute-assignment mismatches, and attribute-type mismatches.

5.2.1 Attribute-assignment mismatch

A mismatch in attribute assignment occurs when two systems both contain knowledge about a concept but differ in the way they assign their attributes to the concept. Consider, a system s_1 that represents that X is a red car as *red_car(X)* and a system s_2 that represents this information as $red(X) \wedge car(X)$. A mapping between these two systems could be done by two simple mapping functions *mf* and *mf`*, such as (*Ont(a)* denotes the ontology of system *a*):

$$mf\,(Ont(s_1){:}\ red\_car(X)) \Rightarrow Ont(s_2){:}\ red(X) \wedge car(X)$$
$$mf`(Ont(s_2){:}\ red(X) \wedge car(X)) \Rightarrow Ont(s_1){:}\ red\_car(X)$$

Another example, derived from Ont-C and Ont-L, concerns the representation of the duration of module registrations. In Ont-C the duration is - implicitly - derivable from the start date (date_start) and the end date (date_end), stored with the concept of a module. In Ont-L the duration is stored as a unit of time in a separate attribute duration of the concept registration. Mutual translation between the two representations is not possible without information loss because the duration attribute does not state what the start and end dates are (*cf.* mismatch in attribute type).

5.2.2 Attribute-type 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 (see also the discussion in section 7). 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'}. 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 kilometers the mapping function obviously requires some basic mathematical operators.

If both system store the same information content then the mapping functions (both ways) will be rather straightforward. However, it is not 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*. This means that the mapping function is one-directional (see section 6 for terminology on mapping functions). If we use a common ontology (knowledge exchange strategy S3, see KP3), this ontology should arguably have the highest granularity level. However, this would not completely solve the problem. Suppose the common ontology has the highest granularity level for values, say, it distinguishes ratios. Now, if a system s_1 that stores marks as 1, 2.1, 2.2 etc., wants to communicate - over the network and thus via the common ontology - with a system s_2 that distinguishes percentages etc., then we've got the problem that we have to 'add' information to translate the information of system s_1 into the common ontology. The reason why we have to add information is that marks cannot unequivocally be translated into ratios. For instance, 1 may correspond to ratio 100/100 or a ratio 25/25, or perhaps even (rounded) non-exact ratios, such as 20/22. We note that if we would make the common ontology so as to have the lowest granularity level we would obviously lose information in the mappings.

5.3 Definition mismatch

If two systems contain different definitions while using either the same term *T* or the same definiens *D*, or while attempting to define the same concept *C* there is a definition mismatch.

5.3.1 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 the definiens.

5.3.2 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 cannot be exchanged without a (non-trivial) mapping function. The solution to such mismatches is to rename one or both concepts, or, to link each of them to their respective ontologies. For instance, Ont-C and Ont-L both use the term module. We could rename the terms to Ont-L-module and Ont-C-module, respectively (alternatively, we could link them to their respective ontologies, say Ont-L:module and Ont-C:module. This solution is suggested, for instance, by Wiederhold (1994)).

5.3.3 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 formal representations (as defined by *D* and *T*) should be kept distinct in a cooperative context. This mismatch type can be resolved by renaming the expressions, or by linking the expressions onto their respective ontologies. For instance, Ont-C and Ont-L both use the term name (definiens: string) to refer to the name of a module. We could, for instance, rename these terms to Ont-C-name and Ont-L-name, respectively. Here, the mapping function ensures that these entities are kept distinct.

5.3.4 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. Either, it can be argued that the terms refer to the same concept *C* for which reason the terms *T* and *T`* should be mapped onto each other, or, it can be argued that the definiens *D* and *D`* are different for which reason there should be no mapping of the terms (possibly the definitions can co-exist in a cooperative context). Which solution is more adequate cannot be stated in general terms, it depends on the relative importance of the differences between *D* and *D`*.

If we consider the TD mismatch between module_desc (Ont-C) and module (Ont-L) we would map the terms onto each other and part of the attributes despite the fact that not all attributes of Ont-L (Ont-C) can be mapped onto attributes of Ont-C (Ont-L).

For the TD mismatch between mark (Ont-C) and student_mark (Ont-L) a similar consideration holds. Both mark entities are intended to model the same concepts although they do not have the same attributes. An instance of mark should be mapped onto an instance of student_mark (and *vice versa*).

5.3.5 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 can be mapped onto each other. The mapping functions are straightforward and bi-directional. For instance, Ont-C refers to the university modules with the term number, and Ont-L refers to this concept with the term code. Note, that their definiens are the same as they are both defined as strings. An example of two mapping functions (both ways) is:

$$mf\,(Ont\text{-}C{:}\ number(X)) \Rightarrow Ont\text{-}L{:}\ code(X)$$
$$mf`(Ont\text{-}L{:}\ code(X)) \Rightarrow Ont\text{-}C{:}\ number(X)$$

5.3.6 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 practice not the same. Hence, in a cooperative context, the terms should be kept distinct. A solution would be to rename the terms, or, to link the terms to their respective ontologies. 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 the terms should remain as they are. A trivial mapping function (*i.e.,* one that links the same terms in different systems) suffices.

If we consider the D mismatch between Ont-C and Ont-L in relation to the definition of degree we would map both degrees onto each other. An instance of a degree in Ont-C should be mapped onto an instance of a degree in Ont-L.

6. Ontology relations: terminology

In the previous sections we investigated the possibilities to resolve ontological mismatches between two communicating systems. In this section we come to a more precise definition of terminology that is useful to describe the relation between ontologies.

An ontology is a tuple that consists of the following five sets: classes, relations, functions, instances, and atoms. This definition satisfies the definition of an ontology in the Knowledge Sharing Effort project (*e.g.,* Gruber, 1995). An ontology is sub ontology of another ontology if it contains a subset of the elements of the latter ontology. Also, two ontologies overlap if there are common elements in either of the five sets (we note that an ontology overlaps with all its sub ontologies).

Definition 1: *ontology*

An ontology is a tuple $O = <C, R, F, I, A>$ in which $C = \{c_1,...,c_n\}$ $(n > 0)$ is a set of classes, $R = \{r_1,...,r_m\}$ $(m \geq 0)$ is a set of relations, $F = \{f_1,...,f_k\}$ $(k \geq 0)$ is a set of functions, $I = \{i_1,...,i_l\}$ $(l \geq 0)$ is a set of instances, and $A = \{a_1,...,a_p\}$ $(p \geq 0)$ is a set of axioms.

Definition 2: *sub ontology*

Ontology $O_1 = <C_1, R_1, F_1, I_1, A_1>$ is a sub ontology of ontology $O_2 = <C_2, R_2, F_2, I_2, A_2>$ if $C_1 \subseteq C_2$, and $R_1 \subseteq R_2$, and $F_1 \subseteq F_2$, and $I_1 \subseteq I_2$, and $A_1 \subseteq A_2$.

Definition 3: *overlap*

Two ontologies $O_1 = <C_1, R_1, F_1, I_1, A_1>$ and $O_2 = <C_2, R_2, F_2, I_2, A_2>$ overlap if either $C_1 \cap C_2 \neq \emptyset$, or $R_1 \cap R_2 \neq \emptyset$, or $F_1 \cap F_2 \neq \emptyset$, or $I_1 \cap I_2 \neq \emptyset$, or if $A_1 \cap A_2 \neq \emptyset$.

To map an expression in one ontology onto an expression in another ontology we define a mapping function (an expression is a well-formed formula in some formal language). If we have a mapping function for each possible ontology expression, we have an ontology mapping.

Definition 4: *mapping function*

If e_1 is an expression in ontology O_1 (shortly: $O_1:e_1$), and e_2 is an expression in ontology O_2, then $mf(O_1:e_1)$

ERROR: undefined
OFFENDING COMMAND: sc

STACK:

0
0
0
[1.2 0.0 0.0 -1.2 140.0 8290.0]