# Mappings Creation Using a View Based Approach*

Adrian Mocan, Emilia Cimpian

Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway, Ireland
adrian.mocan@deri.org, emilia.cimpian@deri.org

**Abstract.** Solving the heterogeneity problems between semantically enriched data can only be done by having accurate alignments between the underlying ontologies. To obtain 100% accuracy of these alignments the human user (i.e. the domain expert) has to be kept in the loop in order to validate the mappings part of these alignments. Since creating mappings between ontologies in a manual fashion can be an error prone and time consuming task, our aim is to provide semi-automatic mechanisms that reduce the human effort to simple validations and choices. Furthermore we propose a mechanism for transforming domain experts inputs placed in graphical interface in formal representations of the semantic relationships between ontologies. The domain expert can choose between different views, each of them displaying certain relationships and entities in the ontologies, used for generating different types of mappings. At each step suggestion algorithms propose possible solutions for creating new mappings.

## 1 Introduction

Ontology mappings have become a prerequisite in solving data heterogeneity problems in the context of Semantic Web and Semantic Web Services. Manual, semi-automatic or automatic approaches have as output a set of so called *mappings*, expression of the semantic relationships existing between the analyzed ontologies. Accordingly, the mappings might be 100% accurate (generated by manual and semi-automatic methods) or could have a lower accuracy (usually generated by automatic methods).

In this paper we present a semi-automatic way of deriving the semantic relationships existing between two ontologies (i.e. mappings) and expressing these relationships in a formalized form. We especially emphasize the step in the mapping process that transforms the domain expert inputs provided by using a graphical interface, in formal representation of mappings. By this, the human user is abstracted from the peculiarities of a specific formalism and they can fully focus on the problem to be solved. The proposed tool offers a set of features meant to reduce the human user efforts during the mapping process from a laborious and error prone task to simple choices and validations. This is done by including a set of mechanisms and strategies to guide the domain expert during the entire mapping process and to suggest potential relationships between the two ontologies. Different views on the ontologies to be mapped can be activated, each of

* This material is based upon works supported by the Science Foundation Ireland under Grant No. SFI/02/CE1/I131.

them focusing on certain ontology entities and their relationships. By switching views different types of mappings can be created using the same principles.

The data mediation prototype we propose is part of Web Service Execution Environment (WSMX) [7], a framework capable of dynamic discovery, selection, mediation and invocation of Semantic Web Services. WSMX has as conceptual model Web Service Modeling Ontology (WSMO) [3] and as a consequence, in our approach, the ontologies to mediate between are WSMO ontologies (i.e. ontologies that conform to WSMO conceptual model for ontologies).

This paper is structured as follows: Section 2 gives a short motivation and describes the context in which this mediator was developed; Section 3 presents the strategies and mechanisms behind the design-time tool we propose; Section 4 presents an example that shows how our approach can be applied and how the generated mapping rules look like; Section 5 briefly describes two of the existing frameworks that relate to our work and Section 6 concludes the paper and indicates some of the future work.

## 2 Motivation

In the Semantic Web Services context it is mandatory for the exchanged data to be semantically described by using ontologies. Furthermore, alignments between different ontologies used for modelling the same domain have to be provided in order to enable the inter-operation of various parties using these ontologies. Such an alignment has to contain a set of (bidirectional) mappings that can be applied on the input data (source data) to produce the corresponding data in terms of the ontologies used by the target party. If the interchanged data are part of a business process the necessity of 100% accuracy[1] of mappings is obvious. This leads us to the immediate consequence that the domain expert has to be kept in the loop to validate these alignments and to assure the 100% accuracy.

The ontology mapping tool developed in WSMX is able to offer support to the domain experts in their work, to reduce the amount of effort required and to create a formal representation of the semantic relationships captured between the source and target ontologies. The semantic relationships are expressed as mappings in the abstract mapping language proposed by [1] and stored in a persistent storage. These mappings are to be used during the run-time when the actual transformations of the interchanged data is performed. As the abstract mapping language doesn't associate any formal semantics to mappings, a grounding to a concrete language has to be provided. More details can be found in [5].

## 3 Mappings Creation - A View Based Approach

The mapping creation process represents one of the most important phases in a mediator system. It is a design time process and it is well known that in order to obtain

[1] By the *accuracy* we don't necessary understand in this work the correctness of the mappings. Therefore, we consider that the mappings are accurate if they match the human user inputs, i.e. domain experts inputs are 100% accurate.

high accuracy of the mappings the human user has to be present in this process. We believe that by offering a set of strategies and methodologies for creating these mappings, we can reduce this error prone and laborious process from a manual task to truly semiautomatic one.

In our view, the mapping process (i.e. the design time phase of the mediation process) basically requires three types of actions from the domain expert:

**Browse the ontologies** The domain expert has to discover the ontology elements they are interested in. This step involves different views on the input ontologies, and strategies for reducing the amount of information to be processed by the human user (e.g. contexts based browsing).

**Identify the similarities** This step involves the identification of semantic relationships between the entities that are of interest in the two ontologies. For doing this the human user can make use of the suggestions offered by a set of lexical and structural algorithms for determining semantic relationships.

**Create the mappings** This last step involves the capturing of the semantic relationships by mappings. This means that the domain expert has to take the proper actions in order to capture the semantic relationships in the mapping language statements or maybe in predefined mapping patterns [2].

We propose a way of tackling the existing gap between the identified semantic relationships of the two ontologies and the mapping language (in our case a logical language) statements that capture these relationships. Mapping patterns can fill this gap only partially, the necessary steps from the semantic relationships identified in graphical tools to these patterns remaining uncovered.

Our approach describes how the input ontologies can be browsed by using different views, how the same ontological entities can play different roles in different views and how certain algorithms can be applied on these roles. We identified a set of views that can play an important role in the mapping creation process: *PartOf* view, *InstanceOf* view, *RelatedBy* view. Each of them will be described in details in this paper.

### 3.1 Terminology and General Strategies

We identify a set of general strategies that can be applied no matter what views are used for browsing the ontologies. Before describing these strategies we have to define several notions that will be used from now on:

**Views** A View presents a subset of the ontology entities (e.g. concepts, attributes, relations, instances) and the relationships existing between them. The views can be seen as vertical subsets of the ontology – all the entities and relationships of a specific type (dictated by each particular view) from the ontology are taken in consideration. Usually the view used for browsing the source ontology (source view) and the view used for browsing the target ontology (target view) have the same type but there could be cases when different view types are used for source and target.

**Roles** In each of the views there is a predefined number of roles the ontology entities can have. In general, particular roles are fulfilled by different ontology entities in different views and in each of the strategies and algorithms described we refer to

roles rather than ontology entities. The roles that can be identified in a view are: *Compound Item*, *Primitive Item*, *Description Item*.

**Compound Item** A *Compound Item* contains at least one description associated with it. For example in the *PartOf* view a compound item would be a concept that has at least one attribute.

**Primitive Item** A *Primitive Item* doesn't have any description associated. For example in the *PartOf* view all data types play the role of a primitive item.

**Description Item** A *Description Item* links a Compound Item with other Compound or Primitive Items. By this we can define as *Successor* of a Description Item, the Compound or Primitive Item it links to (indicated by → in Figure 1).

Figure 1 presents an abstract representation of a view and the main elements it consists of.

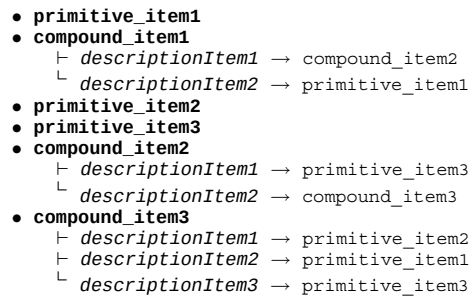


**Fig. 1.** Abstract View

All the algorithms and the strategies we propose are designed to be applicable to any view that meets these abstract specifications. By associating different ontological entities with the roles presented above, different views are obtained, the results being interpreted differently in respect with each particular view. One of the most important advantages of this approach is that these algorithms are immediately reflected in the graphical interface as unique mechanisms that ca be applied no matter of the view used.

**Decomposition Algorithm** The decomposition algorithm is one one of the most important algorithms in our approach and it is used to offer guidance to the domain expert in the mapping process and to compute the structural factor as part of the suggestions algorithms (described later on in this section). By decomposition we expose the descriptions of a compound item and make them available to the mapping process. That is, the decomposition algorithm can be applied on description items and it returns the description items (if any) for the successors of that particular description items. An overview of this algorithm is presented below: The implementation of *isCompound*, *getDescriptions*, *getSuccessor*, and *createLoop* differ from one view to another – for example, the cases when loops are encountered (i.e. the algorithm will not terminate) have to be addressed for each view in particular.

**Table 1.** Decomposition Algorithm

```
decompose(Collection collectionOfItems){
  Collection result;
  for each item in collectionOfItems do {
    if isCompound(item)
      Collection itemsDescriptions = getDescriptions(item);
        for each description in itemsDescriptions {
          Item successorItem = getSuccessor(description);
          if (not createLoop(succesorItem))
            result = result + successorItem;
        }
  }
  return result;
}
```

**Mapping Contexts** During the mediation process not all the information available in the ontology is of interest for each particular phase of the mapping process. A *mapping context* represents a subset of a view and presents only the relevant information for the current step of the mapping process. The notion of mapping context goes hand in hand with the decomposition algorithm as a mapping context is updated by applying this algorithm on a set of items. Thereby, by applying it recursively and updating the corresponding mapping context, the domain expert is guided through the mapping process until all the items from the initial context are mapped. A mapping context can be seen as a horizontal subset of an ontology[2].

Please note that when updating mapping contexts the input of the human user has to be taken in consideration: the domain expert has to choose the source and the target items on which the decomposition process has to be applied. Of course, this choice can be done in a semi-automatic manner, the system suggesting the most probable source-target combinations to be further explored. Depending on the results returned by applying the decomposition algorithm on the source and on the target items respectively, four situations might me encountered:

- *Both sides decomposition.* For both the source and the target items the decomposition algorithm returned a non empty set of items. As a consequence both the source and the target mapping contexts are updated.
- *One side decomposition - Source decomposition.* Only for the source items the decomposition returned a non empty set of items. This means that only the source mapping context is updated while the target mapping context remains unchanged.
- *One side decomposition - Target decomposition.* This is symmetric with the previous case. Only the target mapping context is updated, the source mapping context remaining unchanged.
- *No decomposition.* Successors were found neither for the source nor for the target, so no mapping contexts can be updated. Usually this ends the decomposition and the mapping process for the current branches in the source and target views.

[2] As the ontologies are browsed using views, the context becomes a horizontal subset of a view

**Suggestion Algorithms** The suggestion algorithms are used for helping the domain expert in taking decisions during the mapping process, regarding the possible semantic relationships between source and target items in the current mapping context. We propose a combination of two types of such algorithms: the first one being the lexical based algorithms while the second type being the structural algorithms that consider the description items in their computations.

As a result, for each pair of items we compute a so called *eligibility factor* (EF), which indicates the degree of similarity between the two items: the smallest value (0) means that the two items are completely different, while the greatest value (1) indicates that the two items are similar. For dealing with the values between 0 and 1 a threshold value is used: the values lower than this value indicate different items and values greater than this value indicate similar elements. Setting a lower threshold assures a greater number of suggestions, while a higher value for the threshold restricts the number of suggestion to a smaller subset. The EF is computed as an weighted average between a *structural factor* (SF), referring to the structural properties and a *lexical factor* (LF), referring to the lexical relationships determined for a given pair of items. The SF of two items is recursively determined by calculating the EF for their descriptions and for the successors of their descriptions. As mapped items have the EF equal with 1, is interesting to observe that the suggestions become more accurate the further we advance in the mapping process.

The weights can be chosen based on the characteristics of the ontologies to be mapped. For example when mapping between ontologies developed in different languages the weight of LF should be close to 0 in contrast with the case when mapping between ontology developed in the same working groups or institutions (the usage of similar names for related terms is more likely to happen) .

Even if the structural factor is computed using the decomposition algorithm, the actual heuristics used are dependent on the specific views where it is applied. In a similar manner the current views determine the weight for the structural and lexical factors as well as the exact features of the items to be used in computations.

**Bottom-up vs Top-Down Approach** Considering the algorithms and methods described above two possible approaches regarding ontology mapping can be differentiated: bottom-up and top-down approaches.

The bottom-up approach means that the mapping process starts with the mappings of the primitive items (if possible) and than continues with items having more and more complex descriptions. By this the pairs of primitive items act like a minimal, agreed upon set of mappings between the two ontologies, and starting from this minimal set more complex relationships could be gradually discovered. This approach is useful when a complete alignment of the two ontologies is desired.

The top-down approach implies that the mapping process starts directly with mappings of compound items and it is usually adopted when a concrete heterogeneity problem has to be resolved. That means that the domain expert is interested only in resolving particular items mismatches and not in fully aligning the input ontologies. The decomposition algorithm and the mapping contexts it updates will help the user to identify

all the relationships that can be captured by using a specific type of view and that are relevant to the problems to be solved.

In the same way as for the other algorithms, the applicability and advantages/disadvantages of each of these approaches depends on the type of view used.

**Abstract Mapping Language** The scope of the design-time environment presented in this paper is to produce formal representations of the semantic relationships identified/validated by the domain expert using a graphical tool. We chose to express these relationships as mappings in the language proposed in [1]. It is an abstract mapping language which does not commit to any existing ontology representation languages, thereby a formal semantic has to be associated with it and to ground it to a concrete language. Part of our work was to provide a grounding to Flora2[3] but for space reasons we are not discussing it in this paper (a full description can be found in [5]). From the same reasons we provide only a brief listing of some of the abstract mapping language statements:

- *classMapping* - By using this statement, mappings between classes in the source and the target ontologies are specified. Such a statement can be conditioned by class conditions (*attributeValueConditions*, *attribuiteTypeConditions*, *attributeOccurenceConditions*).
- *attributeMapping* - Specifies mappings between attributes in the source and target ontologies. This statements usually appears together with classMappings and can be conditioned by attribute conditions (*valueConditions*, *typeConditions*)
- *classAttributeMapping* - It specifies mappings between a class or an attribute (or the other way around) and it can be conditioned by both class conditions and attribute conditions.
- *instanceMapping* - It states a mapping between two individuals, one from the source and the other from the target.

In the next sections we illustrate how these mapping language statements are generated during design time by using our view based approach.

### 3.2 *PartOf* View

The *PartOf* is probably the most popular view on the ontologies to be aligned. The roles of *Primitive items* are taken by the *primitive concepts* (i.e. data types) while the roles of *Compound items* are taken by *concepts* described by at least one attribute. The *descriptions* are represented by *attributes* and naturally, the *successor* of a description is the *range* of that attribute. As shown in Figure 2 the successor of a description in this view (i.e. the range of an attribute) can be either a primitive concept or a compound concept.

Using this view we can create the following set of mappings:

**Primitive Concept to Primitive Concept mapping.** Such a mapping generates a *classMapping* statement in the abstract mapping language.

[3] Available at http://flora.sourceforge.net

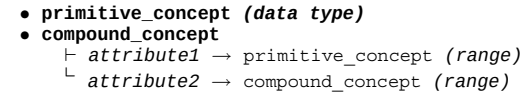


**Fig. 2.** Elements of *PartOf* View

**Compound Concept to Compound Concept mapping.** This mapping generates a *classMapping* statement in the abstract mapping language corresponding to the two compound concepts and triggers the decomposition mechanism, followed by a set of mappings between the attributes of these compound concepts, respectively. Such mappings between attributes are described below.

**Attribute to Attribute mapping.** There are four cases that can be encountered in this situations, generated by the two types of concepts an attribute can have as range: primitive concept or compound concept (i.e. primitive range or compound range).

- *Primitive range* on the source and *primitive range* on the target.
  An *attributeMapping* is generated in the abstract mapping language followed by a mapping between two primitive concepts.
- *Primitive range* on the source and *compound range* on the target.
  This case generates in the abstract mapping language a *classAttributeMapping* between the owner of the source attribute and the target attribute, followed by a mapping between two compound concepts: the owner of the source attribute and the range of the target attribute.
- *Compound range* on the source and *primitive range* on the target.
  This case is symmetric with the one presented above and it generates a *classAttributeMapping* in the abstract mapping language (actually this is an *attributeClassMapping* but there is only one statement in the language for both situations) and leads to a compound concept to compound concept mapping as well.
- *Compound range* on the source and *compound range* on the target.
  An *attributeMapping* is generated in the abstract mapping language followed by a mapping between the two compound concepts.

**Primitive Concept to Compound Concept mapping.** The *PartOf* view does not allow this type of mappings. Such mappings that might seem necessary initially, are covered by considering a compound concept from the source that has (or inherits) an attribute pointing to the primitive concept and mapping it with the compound target concept.

**Compound Concept to Primitive Concept mapping.** This is a situation similar to the one above and the *PartOf* view does not allow this type of mapping. The rational behind these restrictions is that such combinations would generate artificial mappings (with no semantics) between primitive concepts and all the compound concepts that refer, by means of their attributes, to these primitive concepts. For example, any compound concept that has an attribute with the range *String* could be mapped with *String*.

### 3.3 *InstanceOf* View

During the modeling process a set of instances can be used to properly capture some of the features of the domain. This is the case for enumeration sets, containing for example geographical locations, categories or even the allowed values for certain data-types (e.g. true and false for boolean). In the *PartOf* view these instances are not visible, however in the *InstanceOf* view the primitive items' role is taken by such instances (we call them *primitive instances*). By using these primitive instances more complex instances (*compound instances*) could be created, that is, instances of compound concepts whose attributes have ranges for which primitive or compound instances already exist or can be created. The compound instances play the role of compound items in this view (see Figure 3). The descriptions for the compound instances are represented by the attributes and attribute values corresponding to the compound instances. The attribute values are in fact the successors of compound items descriptions.

Additionally in this view we have to consider the rest of the concepts, for which no compound instances can be created. They might be either primitive concepts (they have no attributes at all) or compound concepts (none of their attributes has a range for which a primitive instance exists or a compound instance can be created) as identified in the *PartOf* view. They will play in this view the role of primitive items and all of them will be called *primitive concepts*.

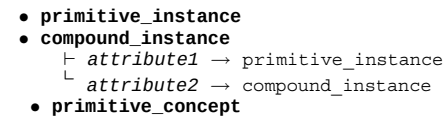


**Fig. 3.** Elements of *InstanceOf* View

*InstanceOf* view is used for creating conditional mappings and almost all the cases presented in the *PartOf* view occur in this view as well but with the difference that conditions are associated to mappings:

**Primitive Instance to Primitive Instance mapping.** This mapping generates an *instanceMapping* statement in the abstract mapping language.

**Primitive Instance to Compound Instance mapping.** Mappings between a primitive and a compound instance are not allowed in the *InstanceOf* view from similar reasons as for *Primitive Concept to Compound Concept mapping*in the *PartOf* view.

**Compound Instance to Primitive Instance mapping.** The same restriction applies as above.

**Compound Instance to Compound Instance mapping.** This mapping generates a *classMapping* statement in the abstract mapping language corresponding to the two compound concepts that are instantiated by the compound instances and triggers the decomposition mechanism, followed by a set of mappings between the attribute values of these compound instances, respectively. Such mappings between attributes' values are described below.

**Attribute value to Attribute value mapping.** There are four cases that can be encountered in this situation, generated by the two types of instances an attribute can have as value: primitive instance or compound instance (i.e. primitive instance range or compound instance range).

- *Primitive instance range* on source and *primitive instance range* on target.
  An *attributeMapping* is generated in the abstract mapping language conditioned by two *attributeValueConditions* - one for the source and the other one for the target attribute.
- *Primitive instance range* on source and *compound instance range* on target.
  This case generates in the abstract mapping language a *classAttributeMapping* between the owners of the source attribute and target attribute followed by a mapping between two compound instances: the owner of the source attribute and the range of the target attribute. In addition a *typeCondition* on the target attribute is applied.
- *Compound instance range* on source and *primitive instance range* on target.
  This case is symmetric with the one presented above and it generates a *classAttributeMapping* in the abstract mapping language and leads to a compound instance to compound instance mapping as well. In addition a *typeCondition* on the source attribute is applied.
- *Compound instance range* on source and *compound instance range* on target.
  An *attributeMapping* and two *typeConditions* one for the source and the other one for the target attribute, are generated in the abstract mapping language followed by a mapping between two compound instances.

In this view, no decomposition process can be applied on the primitive concepts and they can participate in the following types of mappings:

**Primitive Instance to Primitive Concept mapping.** This type of mapping is not allowed as the scope of primitive instances is to set conditions when mapping between other ontological entities. In this case, most probably a Compound Instance (created based on the initial primitive instance) with a Primitive Concept mapping should be considered.

**Primitive Concept to Primitive Instance.** The above restriction applies here as well.

**Primitive Concept to Compound Instance mapping.** This type of mapping will trigger decomposition on the right side and will generate a *classMapping* statement in the abstract mapping language. After decomposition, a set of mappings between the primitive concept and the attribute values of the compound instances can follow.

**Primitive Concept to Attribute value mapping.** Depending on the range of a particular attribute value (primitive or compound instance) we have two cases:

- *Primitive concept* on source and *primitive instance range* on target.
  An *attributeValueCondition* is added to the mapping between the primitive concept and the concept that owns the attribute (used in the above attribute value).
- *Primitive concept* on source and *compound instance range* on target.
  A *typeCondition* is added to the mapping between the primitive concept and the concept that owns the attribute (used in the above attribute value). After this *primitive concept to attribute value* mappings are to be considered.

### 3.4 *RelatedBy* View

Another interesting view to consider is the one where *attributes* play the roles of items. Each item has two fixed descriptions, one called *hasDomain* and the other called *hasRange*, having as successor the domain and the range of the attribute, respectively. Because of these two fixed descriptions, all the items are *compound items*. In addition, applying the decomposition algorithm to the descriptions and their successors will trigger a change of the view type. As the successors are concepts, by decomposition the view will switch from *RelatedBy* view (see Figure 4) to the *PartOf* view.

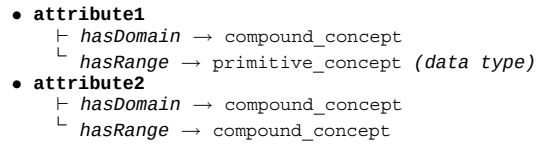


**Fig. 4.** Elements of *RelatedBy* View

We have two interesting types of mappings that can be done using this view: mappings between two *RelatedBy* views, and mappings between a *RelatedBy* view and a *PartOf* view (for the source and target respectively). For the first case we simply have:

– **(Compound) Attribute to (Compound) Attribute Mapping.** This is a classical mapping between two attributes, which will be followed after applying the decomposition algorithm by mappings between their domains and ranges. Such mappings can be done using the *PartOf* view as well, with only one difference: when created using the *RelatedBy* view, inverse attributes can be mapped, simply by mapping *hasDomain* from the source with the *hasRange* description from the target and *hasRange* from the source with *hasDomain* from the target. One has to keep in mind that the decomposition will trigger a view switching (from *RelatedBy* to *PartOf* view) so these mappings are affected by the restriction of the *PartOf* view (i.e. no mappings between primitive and compound concepts or vice versa are allowed). This type of mappings generates an *attributeMapping* and a pair of *classMappings* statements in mapping language.

When mapping between *RelatedBy* and *PartOf* view (and vice versa) we can have the following types of mappings:

– **(Compound) Attribute** (from *RelatedBy*) **to Compound Concept** (from *PartOf*). This mapping can be followed by any mappings between *hasDomain* and *hasRange* descriptions and descriptions of the compound concept from the target (i.e. attributes) as long as the mappings between successors conform to the restrictions on mappings in the *PartOf* view. A *classAttribute* mapping will be generated, followed by one or more pairs of *classAttributeMapping* and *classMapping*.
– **Compound Concept** (from *PartOf*) **to (Compound) Attribute** (from *RelatedBy*). This is a similar type of mapping as the one presented above. A *classAttribute* mapping is created, followed by one or more pairs of *classAttributeMapping* and *classMapping*.

In the next section we exemplify how a small ontology fragment is captured through the three types of views presented above.

## 4 Example

Let's consider the two fragments of ontology in Table 2 and try creating different types of mappings by using different types of views. For each pair of views we will present the abstract mappings generated from the graphical tool together with the grounding from this abstract representation to Flora-2.

Throughout the example we will represent the attributes in the following form in order to make the mappings self-explanatory:

```
[(attribute_owner) attribute_owner.attribute name => attribute_range]
```

The concepts will be referred simply by their names. The mapping process can

**Table 2.** Source and a target ontology fragments to be mapped

```
concept person
     name ofType xsd:string
     age ofType xsd:integer
     hasGender ofType gender
     hasChild ofType person
     marriedTo ofType person

concept gender
     range ofType xsd:string

instance male memberOf gender
     range hasValue "male"

instance female memberOf gender
     range hasValue "female"
```

```
concept human
     name ofType xsd:string
     age ofType xsd:integer
     noOfChildren ofType xsd:integer

concept marriage
     hasParticipant ofType human
     date ofType xsd:date

concept man subConceptOf human

concept women subConceptOf human
```

start by creating mappings using the *PartOf* view for both ontology fragments (see Figure 5). The mappings generate the abstract mapping language statements illustrated in Table 3. In order to use these mappings in concrete mediation scenarios a grounding mechanism is necessary to associate formal semantics to them. Table 4 shows how these mappings can be grounded to Flora-2. The dotted lines in Figure 5 shows mappings that are automatically created due to the inheritance relation existing between *human* and *man* and *human* and *woman* concepts.

Another set of mappings can be created by using *InstanceOf* views (see Figure 6). These mappings create a set of mapping rules and associate conditions to them, meaning that a particular mapping is valid only if the associated conditions hold.

Table 5 shows these mappings in the abstract mapping language while Table 6 contains the same mappings grounded to Flora-2.

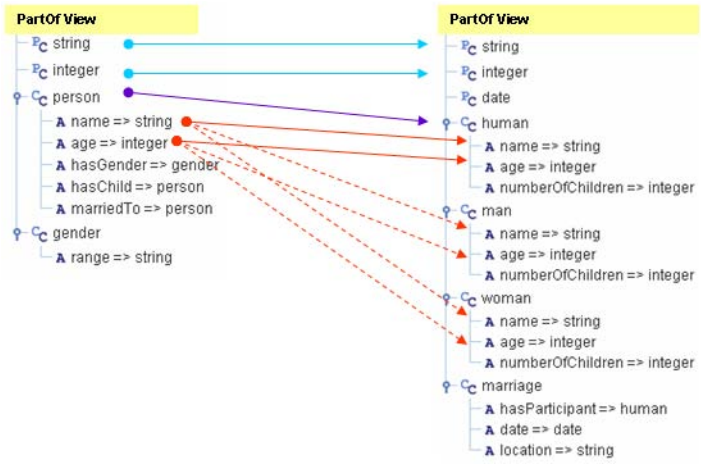


**Fig. 5.** Example of mappings between two *PartOf* Views

**Table 3.** Abstract mapping language statements generated by using *PartOf* views

```
Mapping(id000001
  classMapping( one-way
    string  string))
Mapping(id000002
  classMapping( one-way
    integer integer))
Mapping(id000003
  classMapping( one-way
    person  human))
Mapping(id000004
  attributeMapping( one-way
    [(person) person.name => string]    [(human) human.name => string]))
Mapping(id000005
  attributeMapping( one-way
    [(person) person.age => integer]    [(human) human.age => integer]))
```

**Table 4.** Flora-2 statements generated from the abstract mappings in Table 3

```
mediated(X_2, string):string :- X_2:string.
mediated(X_3, integer):integer :- X_3:integer.
mediated(X_4, human):human :- X_4:person.
mediated(X_5, human)[human.name -> Y_6] :- X_5[person.name -> Y_6].
mediated(X_7, human)[human.age -> Y_8] :- X_7[person.age -> Y_8].
```

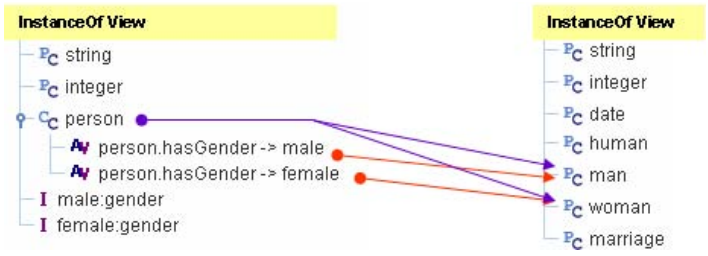


**Fig. 6.** Example of mappings between two *InstanceOf* Views

**Table 5.** Abstract mapping language statements generated by using *InstanceOf* views

```
Mapping(id000011
  classMapping( one-way
    person  man)
    attributeValueCondition(person [(person) person.hasGender => string] male)
Mapping(id000012
  classMapping( one-way
    person  women)
    attributeValueCondition(person [(person) person.hasGender => string] women)
```

**Table 6.** Flora-2 statements generated from the abstract mappings in Table 5

```
mediated(X_9, man):man :- X_9:person, X_9[hasGender -> male].
mediated(X_10, woman):woman :- X_10:person, X_10[hasGender -> female].
```

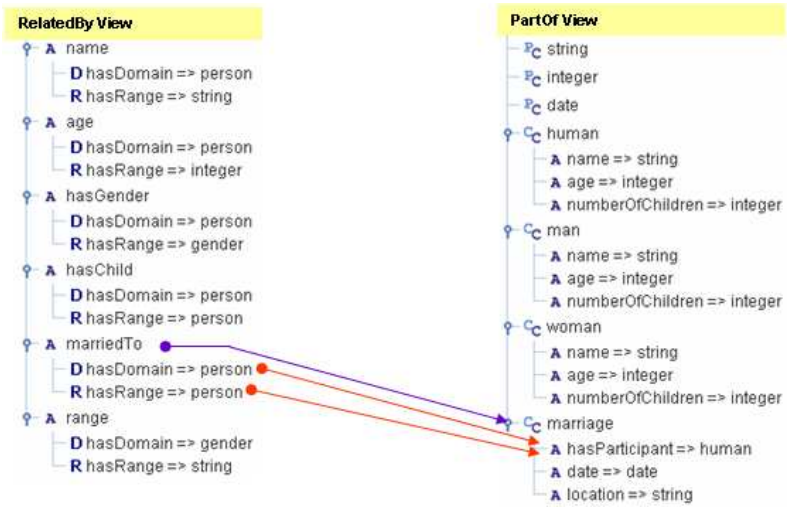


**Fig. 7.** Example of mappings between two *RelatedBy* Views

Another set of mappings can be created by using *RelatedBy* and *PartOf* views as depicted in Figure 7, for the *marriedTo* relation.

These mappings generate two one *classAttributeMapping* and one *attributeClassMapping* denoted in the abstract mapping language by the same statement, *classAttributeMapping* (see Table 7). Table 8 depicts the grounding of the abstract mappings in Flora-2.

**Table 7.** Abstract mapping language statements generated by using *RelatedBy* and *InstanceOf* view

```
Mapping(id000021
  classAttributeMapping( one-way
    [(person) person.marriedTo => person]  marriage))
Mapping(id000022
  classAttributeMapping( one-way
    person  [(marriage) marriage.hasParticipant => human]))
```

**Table 8.** Flora-2 statements generated from the abstract mappings in Table 7

```
mediated(X_25, marriage):marriage :- Y_26[person.marriedTo -> Z_27],
                                     mediated(X_25, human):human,
                                     Z_27:human, Y_26:person.
mediated(X_24, marriage)[marriage.hasParticipant -> mediated(Y_23, human)] :-
                                     Y_23:person,
                                     mediated(X_24, marriage):marriage.
```

## 5 Related Work

PROMPT is an algorithm and a tool proposed by Noy and Musen [6], which allows semi-automated ontology merging and alignment. It takes as inputs two ontologies and guides the user through an iterative process for obtaining a merged ontology as an output. This process starts with the identification of the classes with similar names and provides a list with initial matches. Then the following steps are repeated several times: the user selects an action (by choosing a suggestion or by editing the merged ontology directly) and the tool computes new suggestions and determines the eventual conflicts.

MAFRA [4] is a Mapping Framework for Distributed Ontologies, designed to offer support at all stages of the ontology mapping life-cycle. The framework is organized in two dimensions: it contains horizontal and vertical modules. The horizontal modules (*Lift & Normalization*, *Similarity*, *Semantic Bridging*, *Execution*, *Post-processing*) describe fundamental and distinct phases in the mapping process, while the vertical modules (*Evolution*, *Domain Constraints & Background Knowledge*, *Cooperative Consensus Building*, *Graphical User Interface*) run along the entire mapping process interacting with the horizontal modules.

Our approach offers a set of strategies that guides the domain expert through the whole mapping process. By maintaining proper mapping contexts and applying the decomposition process we can ensure that a complete set of mappings are derived for

that specific problem and give a precise meaning to user's actions. Further more, alternating different views on the ontologies to be mapped we can change the meaning of these actions without changing the mapping process. Each of the views we identified can be used to address different types of ontologies mismatches in a natural way and to abstract the human expert from the burdensome of the underlying mappings representation. And finally, we generate a representation of the identified semantic relationships as mappings in an abstract mapping language offering a great flexibility with respect to the mapping context in which these mappings are to be used.

The ideas presented in this paper have been implemented as part of WSMX Ontology Mapping Tool, delivered as plug-in in the Web Service Modeling Toolkit[4].

## 6 Conclusions and Future Work

Our approach proposes a methodology for mapping creation based on the inputs and validations of a domain expert. The resulting mappings have a 100% accuracy with respect to the domain expert inputs. They are represented as statements in an abstract mapping language, leaving open the possibility to create the most suitable grounding for a specific application.

Our future plans include refinements of the strategies and methodologies we introduced as: the identification of more views that can have a relevant role in mapping creations, improvements of the suggestion algorithms (including tuning mechanism) and of course enhancements of the design-time tool towards a more intuitive and user friendly graphical interface.

## References


1. J. de Bruijn, D. Foxvog, and K. Zimmerman. Ontology mediation patterns. Technical report, SEKT Deliverable D4.3.1, 2004.
2. Jos de Bruijn, Douglas Foxvog, and Kerstin Zimmerman. Ontology mediation patterns library. SEKT Project Deliverable D4.3.1, Digital Enterprise Research Institute, University of Innsbruck, 2004.
3. C. Feier, A. Polleres, R. Dumitru, J. Domingue, M. Stollberg, and D. Fensel. Towards intelligent web services: The web service modeling ontology (WSMO). *International Conference on Intelligent Computing (ICIC)*, 2005.
4. A. Maedche, B. Motik, N. Silva, and R. Volz. Mafra - a mapping framework for distributed ontologies. *Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW)*, September 2002.
5. A. Mocan and E. Cimpian. WSMX Data Mediation. Technical report, WSMX Working Draft, `http://www.wsmo.org/TR/d13/d13.3/v0.2/`, March 2005.
6. D. Noy and M. Munsen. Prompt: Algorithm and tool for automated ontology merging and alignment. *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2000.
7. K. Verma, A. Mocan, M. Zaremba, A. Sheth, and J. A. Miller. Linking semantic web service efforts. *Second International Workshop on Semantic and Dynamic Web Processes (SDWP)*, 2005.


[4] Fore more information see `http://sourceforge.net/projects/wsmx/`