A Semantic Web Based Ontology Mapping and Instance Transformation Framework

Borna Jafarpour
NICHE Research group
Computer Science Department, Dalhousie University
Halifax, Canada
borna@cs.dal.ca

Syed Sibte Raza Abidi
NICHE Research group
Computer Science Department, Dalhousie University
Halifax, Canada
Sraza@gmail.com

We present a semantic-based ontology mapping framework that offers instance transformation and discovery of new mapping using reasoning. Our framework comprises an expressive OWL-Full Mapping Representation Ontology (MRO) and a mapping translation method. Ontology mappings are represented in terms of an instantiation of the MRO. We define formal semantics for our ontology mapping representation by translating the ontology mappings in OWL-Full to OWL and SWRL in order to derive new ontology mappings and perform instance transformation using reasoning. We have evaluated the workings of our ontology mapping framework by mapping three ontologies each representing a disease specific Clinical Practice Guideline (CPG) to a general CPG representation ontology. The intent of the mapping is to provide knowledge-driven decision support for the management of patients with multiple diseases.

Keywords—Ontology; Semantic Web; Ontology mapping; Instance Transformation; SWRL, OWL

I. INTRODUCTION

Complex knowledge-centric systems demand the integration of multiple knowledge objects in order to achieve a comprehensive knowledge model. Given the open nature of semantic web, several heterogeneous knowledge models exist for representing the knowledge in any domain area. For instance, in healthcare, there exist variety of knowledge models to model and computerize clinical practice guidelines (CPG)—these models share a range of concepts but differ in the interpretation and specification of these concepts. To develop a holistic knowledge model based on multiple heterogeneous knowledge models, therefore demands the establishment of standardized interoperability specifications and criterion, at both the structural and semantic levels, to achieve the integration of multiple heterogeneous knowledge models.

Lately, ontologies have emerged as expressive knowledge representation formalisms, together with methods to reason over the knowledge. An ontology typically represents a specific aspect of knowledge with varying levels of abstraction and description. To formulate a broader and holistic knowledge model, researchers aim to integrate multiple existing ontologies that demand an interoperability solution that aligns heterogeneous ontologies in keeping with the domain-specific interpretations and constraints surrounding knowledge consistency. A semantic interoperability framework aims to establish explicit and well-defined *mapping* between two ontologies. In practice, ontology mappings methods map the ontology elements between two ontologies based on the similarity of their names, their relations and their shared instances using name-based, structure-based and instance-based approaches respectively [10].

An alternative mapping approach is called semantic-based ontology mapping. This approach has two steps [10]: (i) *anchoring step* in which a number of initial mappings or anchors are created between two ontologies using name, instance or structure based ontology mapping approaches; (ii) *reasoning step* in which a reasoner performs reasoning on the mappings and the mapped ontologies to (a) transform instances between the two ontologies; and (b) improve the existing mappings by discovering new ones based on the formal semantics of the mappings and the mapped ontologies. Typically, proprietary reasoning algorithms [1][4][14], propositional logic [11][12] and Description Logic (DL) [5][6][7][8][9] are used in the reasoning step.

The quality of ontology mapping based on a semantic-based approach is contingent on the ontology mapping representation language's level of expressivity and formal semantics—reasoning over a more expressive ontology can yield more new mappings as opposed to reasoning over a less expressive ontology. Our review of the existing mapping representation languages [1][2][3][4][9][11][12][13][15] and an existing surveys [13] reveal that most of the current ontology mapping languages suffer from lack of expressivity and formal semantics. Lack of formal semantics stops us from using the mappings in a semantic-based ontology mapping approach.

To address the lack of expressivity and formal semantics in ontology mapping languages, in this paper we use semantic web technologies to present a semantic-based ontology mapping approach that entails: (a) a general purpose OWL-Full based *Mapping Representation Ontology* (MRO) that serves as an expressive ontology mapping language that can represent complex mappings such as predefined mapping patterns, conditions, condition satisfaction criteria, variables, structural modifications and mathematical operators. An instance of the MRO represents the mappings between a source and a target ontology; and (b) *translation algorithm* to translate the instantiations of the MRO (which are in OWL-Full and hence undecidable) to OWL-DL or OWL 2 RL + SWRL which is a decidable combination. The translated mappings and the

mapped ontologies are reasoned over to achieve both instance transformation and to discover new mappings. Please note that our approach is not problem-specific and can be used for mapping any two ontologies as long as they are represented in OWL.

We chose to represent mappings in OWL-Full and then translate them to OWL+SWRL instead of using OWL+SWRL directly because of the following reason: (a) The expressivity of MRO being OWL-Full—i.e. using properties and classes as instances—makes the ontology mappings more readable and less verbose—i.e. with fewer triples compared to OWL-DL; (b) It enables us to support conditional mappings and complex condition satisfaction criteria, meta modelling, Boolean operators and converting ontology elements and creating new ones which are not directly supported by either OWL or SWRL. These aspects of ontology mapping are supported by automatic generation of several OWL axioms and SWRL rules that simulate the lacking feature during the translation process; (c) SWRL rules are difficult to write and can easily become undecidable if not written correctly. In our translation algorithm, DL-Safe SWRL rules are generated automatically thus relieving the user about decidability concerns.

In order to evaluate the efficacy of our ontology mapping framework, we instantiated MRO to map three disease-specific CPG ontologies to a general CPG ontology. We then successfully transformed instantiations of the source ontologies to instantiations of the target ontology. The problem being pursued here is to handle comorbidities by integrating two or more disease-specific CPG to manage a patient with multiple simultaneous diseases.

II. Related Work

In this section, we review the existing semantic-based ontology mapping approaches and the existing mapping representation languages.

A. Semantic-Based Ontology Mapping Approaches

These approaches can be categorized based on the reasoning techniques that they use. Literature reports on using proprietary reasoning algorithms [1][4][14], propositional satisfiability solvers [11][12] and description logic reasoners [5][6][7][8][9].

Methodologies that use proprietary reasoning algorithms such as [1][4][14] are not desirable because of the following disadvantages: (a) Because of their proprietary algorithms, they can't benefit from the existing reasoners and a special reasoning engine should be developed in order to perform the reasoning step; (b) Since these engines can only perform reasoning on the mappings and not the ontology representation languages they cannot exploit the internal structure (knowledge) of the ontologies to draw new mappings based on them.

There are semantic-based algorithms that use propositional logic to perform reasoning. In these approaches, a theory is built by conjunction of the axioms from the mapped ontologies. This theory can be constructed by using one of the name, instance or structure based approaches. Then, a matching formula is made for each pair of classes from the mapped ontologies. Afterwards, the validity of the formula is checked by using a propositional satisfiability solver. BerkMin [11] and GRASP [12] are two examples to name. None of these approaches goes beyond finding equivalence, subclass, and complement relationship between classes. We believe that this is due to lack of expressivity in propositional logic for the task of ontology mapping.

Description logic reasoners have also been used in the reasoning step of semantic-based ontology mapping approaches. Two approaches that use description logic to find disjointness, overlap, inclusion and equivalence relations between concepts are reported in [5][7]. Meilicke and colleagues [6] used description logic to debug the mappings by detecting inconsistencies. In a theoretical work [8] it is suggested that description logic can be used for reasoning about the mapping themselves to find containment, minimality, consistency and embedding attributes in them. Therefore, DL has been used for reasoning about the mappings, debugging them and deriving simple mappings (class equivalence, etc.) but no attempt has been made to represent more complex mappings such as value transfer mappings or mathematical computations. We believe that lack of an expressive mapping representation language that formally defines the mapping semantics in DL is limiting the capabilities of DL-based semantic-based mapping methodologies.

There are also approaches such as [1] and [3] that translate the mappings to OWL and SWRL to use OWL reasoners. These methodologies transform the mapping to either OWL or SWRL but not a combination of them. However, we believe that OWL or SWRL cannot be used separately for mapping ontologies unless we need very low levels of expressivity. Therefore, we can conclude that complex mappings are not possible to be transformed using these approaches. Moreover, no explanation or details of the translation process have been provided in this regard.

B. Ontology mapping representation languages

In this section, we review the expressivity levels of the mapping representation languages with formal semantics. We reviewed the literature trying to define the requirements of the mapping representation languages [13][15]. The support for mathematical, Boolean, string and structural modification operators, frequently used mapping patterns, predefined set of relations between ontology elements, variables and the ability to express conditions and condition satisfaction criteria are the most important expressivity requirements identified in these publications.

Many of the existing mapping representation languages such as MAFRA [4], C-OWL [9] and many others [1][4][5][6][8][9][11][12] are only capable of expressing simple relations such as equivalent, disjoint, subclass and super class between ontology classes. A review of 13 of these languages in [13] shows that 61% of all of them are only capable of expressing the equivalence relationship. Only C-OWL has formal semantics that can be used by reasoners in the semantic-based ontology mapping. Even though authors of MARFA claim that they have formal semantics no details are provided in that regard. OWL is more expressive than these

languages as it supports a wide range of predefined relations between classes, properties and instances. It also supports a large number of class and property manipulation operators that can be used towards structural modification. The rest of the desired features described earlier are not supported by OWL. Having formal semantics makes it possible to use OWL in the reasoning step of a semantic-based ontology mapping approach.

An important requirement of these languages is the ability to support variables and to express mathematical, Boolean, date, string computation and comparison and structural modification operators. SWRL is the only language that is able to express a wide list of the necessary functions for mappings that are supported by the concept of built-ins. This language however cannot support Boolean operators, mapping patterns, conditions, qualified cardinality restrictions and some of the property relations and structure modifications operators that are expressible in OWL such as union operator. SWRL also has formal semantics and can be used in semantic-based ontology mapping approaches.

Two expressive mapping languages are discussed in [2] and [3]. The language in [2] supports a wide range of mappings patterns, conditions and variables. However, this language does not support representation of complex condition satisfaction criteria, and mathematical, Boolean, string and date operators. The language in [3] supports a large number predefined set of relations between ontology elements, mapping patterns, ability to express conditions and structural modification operators. Even though some descriptions of the formal semantics of these languages are discussed, enough details for a practical implementation of a semantic-based ontology mapping approach are not provided.

III. Our Ontology Mapping Approach

Our ontology mapping approach entails the following two components: A Mapping Representation Ontology (MRO) in OWL-Full to represent the ontology mapping; and a translation algorithm that transforms an instantiation of the MRO to OWL + SWRL. Our ontology mapping approach is pursued by performing the following three steps:

1. Anchoring (MRO Instantiation): In the first step, initial inter-ontology mappings are created by establishing semantic relations between classes, properties and instances of the mapped ontologies. These mappings can be either created using existing automatic discovery algorithms such as methods based on similarity of names or by a domain expert. Due to complexity of the mappings between ontologies of our domain area, we opted to create the initial mappings manually. Therefore, a mapping between two ontologies is an instantiation of the MRO created by the domain expert. For instance, by instantiating MRO we may indicate that classes *Person* and *Human* from source and target ontologies are equivalent classes. Source and target ontologies are represented by o1: and o2: name spaces in the rest of the paper.

2. Translation to OWL-DL + SWRL: In the next step, we transform the instantiation of MRO to a combination of OWL-DL or OWL2 RL + SWRL depending on the expressivity needs of the mappings. To avoid the possible undecidability as the result of using SWRL rules, only DL-Safe rules [17] are added in the translation process. As an example, the instantiation of MRO that expresses *Human* and *Person* classes are equivalent is translated to:

```
o1:Human owl:equivalentClass o2:Person.
```

3. Reasoning: Finally, we use OWL reasoners to perform reasoning on the translated mappings and the mapped ontologies to improve the mapping by discovering new ones and to perform instance transformation. As an example, The following SWRL rule which is the result of the translation of an instantiation of MRO to SWRL, calculates the Body Mass Index (BMI) of an instance of the class *o1:Person* and assigns it to the class *o2:ObesePerson* if the value of the BMI is greater than 30 and the condition c1 is satisfied.

```
o1:Person(?InstVar), o1:hasHeight(?InstVar,
?HeightVar), o1:hasWeight(?InstVar,
?weightVar),swrlb:divide(?BMIVarVar,
?func1SWRLVar,?HeightVarVar),swrlb:divide(?fun
c1Var,?weightVar,?HeightVar),swrlb:greaterThan
(?BMIVar, 30),SatisfiedCondition(c1)->
o2:ObesePerson(?o1InstVar),
o2:hasBMI(?o1InstVar, ?BMIVar)
```

As a result of reasoning on this rule and source and target ontologies all together, the reasoner infers that an instance of the *Person* class in the source ontology with the weight of 97kg and height of 179cm belongs to the class *o2:ObesePerson* in the target ontology and has the value 32.2 for the property *o2:hasBMI*. In this way, instances of the *o1:Person* class in the source ontology are transformed to instances of the *o2:ObesePerson* class in the target ontology. We have used Pellet as our reasoner since it supports both OWL and SWRL. Any other reasoners with support for OWL and SWRL can be used for this purpose.

IV. Mapping Representation Ontology

In this section, we describe MRO, its classes, properties and instances. In order to easily identify classes, properties and instances in the text, class names are *italicized* and their first letter are Capitalized (e.g. *ClassNameExample*), property names are *italicized* (e.g. *propertyExample*) and instance names are underlined (e.g. instanceExample).

A. Mappings and Relations

In order to represent mappings between instances, properties and classes of source and target ontologies we have created the *Mapping* class. Three types of mappings have been modeled in MRO using the following classes: *RelationalMapping*, *TransformationMapping* and *ValueTransferMapping*.

RelationalMappings express a relation between two ontology elements. *hasSource* and *hasTarget* properties with the domain of *Mapping* and range of OWL:thing are used assign the source and the target elements to a mapping. The *hasRelation* Property with the domain of *RelationalMapping* and the range of *MappingRelation* defines the relation in a relational mapping. Depending if the relation is between two

instances, properties or classes, one of the instances of the *InstanceRelation*, *PropertyRelation* or *ClassRelation* classes is used. In the following example we have used subClassRelation an instance of the *ClassRelation* to map the *o1:Father* class as a subclass to the *o2:MalePerson* class:

```
:m1 a:RelationalMapping;
  :hasSource    o1:Father;
  :hasTarget    o2:MalePerson;
  :hasRelation :subClassRelation
```

TransformationMapping specifies how the source ontology elements need be structurally modified and transformed to elements of the target ontology. Two types of transformation mapping have been modeled: (i) Property to class mapping represented by *PropToClassTransMapping* class. As an example, a property to class transformation mapping transforms the OWL triple `o1:john o1:isMarriedTo o1:jane` to

```
o2:john_jane_marriage a o2:Marriage;
 o2:hasMalePartner o1:john;
 o2:hasFemalePartner o1:merry.
```

As you can see, an instance of the class *o2:Marriage* for each pair of instances connected by the property *o1:isMarriedTo* should be created. The following instantiation of the mapping class represents this mapping from o1 to o2.

```
:m a :PropToClassTransMapping;
   :hasSourceProperty o1:isMarriedTo
   :hasTargetClass o2:Marriage;
   :hasTargetProperty1 o2:hasMalePartner;
   :hasTargetProperty2 o2:hasFemalePartner.
```

Please note the *hasSourceProperty*, *hasTargetClass*, *hasTargetProperty1* and *hasTargetProperty2* properties in this mapping and their purposes.

(ii) Class to property mapping which is the exact opposite of the property to class mapping. This mapping is represented by the *ClassToPropertyTransMapping* class. The following instantiation of the mapping ontology performs the exact opposite transformation in the abovementioned example from o2 to o1:

```
:m a :ClassToPropertyTransMapping;
   :hasTargetProperty o1:isMarriedTo
   :hasSourceClass o2:Marriage;
   :hasSourceProperty1 o2:hasMalePartner;
   :hasSourceProperty2 o2:hasFemalePartner.
```

Please note the *hasTargetProperty*, *hasSourceClass*, *hasSourceProperty1* and *hasSourceProperty2* properties in this mapping and their purposes.

ValueTransferMappings perform mathematical, string and date computation and comparison to find the new value in the target ontology based on the value of the source ontology. An instance of this mapping would be computing the Body Mass Index in the target ontology based on the weight and the height of a person in the source ontology. No relation is assigned to this type of mapping. The *hasFunction* property with the range of *Function* is used to assign the participating functions in data transformation to a mapping.

B. *Variables*

Variables that are represented by the *Variable* class can be used to represent values or a fragment of the ontology and be used as the source or target of the mappings. Fig. 1 shows subclasses of the *Variable* class. It has two subclasses: *ClassVariable*, *InstanceDataVariable*.

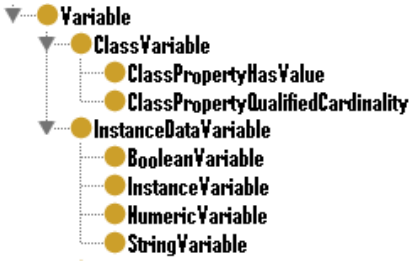

Fig. 1. Subclasses of the Variable Class

1) ClassVariable: This class and its associated properties can be used to represent a class of instances. It has two subclasses: *ClassPropertyHasValue* and *ClassPropertyQualifiedCardinality*. *ClassPropertyHasValue* can be used to create a variable which represents a class whose instances have a specific value for a specific property. For instance, the following class variable represents the students who have taken course math101 for the summer:

```
:cv1 a :ClassPropertyHasValue;
 :classVariableHasClass o1:Student;
 :classVariableHasProperty o1:hasSummerCourse;
 :classPropertyRestrictionHasValue o1:math101.
```

An instance of the *ClassPropertyQualifiedCardinality* class represents instances that have a restriction on the number and type of values that a specific property can have. For instance, we can create a class that represents students who have registered for at least two elective courses:

```
:cv2 a: ClassPropertyQualifiedCardinality;
  :classVariableHasClass o1:Student;
  :classVariableHasProperty o1:hasCourse;
  :classPropertyQCROnClass o1:ElectiveCourse
  :hasCardinalityType :min;
  :hasNumericValueForCardinality "2"^^xsd:int.
```

hasCardinalityType with the range of *Cardinality* represents the cardinality type. Instance of the *Cardinality* class are any, all, min and max.

2) InstanceDataVaraible: They have a similar purpose to data varible in programming languages. They can hold a string, numeric, boolean values or represent an instance of the ontology using sublcasses *StringVariable*, *NumericVariable*, *BooleanVariable* and *InstanceVariable* respectively. In the following example, we create an instance variable which represents all the instances of the *Student* class in the source ontology and a data variable which represents the weight of the student represented by the instance variable:

```
:studentVar a :InstanceVariable.
:weightVar a :NumericVariable.
```

```
:cv1 a :ClassPropertyHasValue;
 :hasInstanceVariable :studentVar;
 :classVariableHasClass o1:Student;
 :classVariableHasProperty o1:hasWeight;
 :classPropertyRestrictionHasValue :weightVar.
```

The value of the *weightVar* variable can be compared with a predefined number and the result can be used to make the decision whether the instance variable *studentVar* belongs to the class *o2:ThinStudent* or *o2:NormalWeightStudent* in the target ontology. In order to perform such a mapping we need to be able to define mathematical functions.

C. Functions and Operators

Expressivity of a mapping representation language is highly dependent on its support for representation of Boolean, mathematical, string, date and instance comparison and computations.

The *Function* class is the smallest entity that can be used for computation in our mapping ontology. Each function accepts an operator, a set of input variables and generates an output. A function has at most two inputs that are assigned to it by *hasInput1* and *hasInput2* properties with the domain of *Function* and range of *Variable*. Outputs of functions are assigned to them by the object property *hasOutput* with the range of *Variable* class. The operator of a function is assigned to it by the *hasOperator* property with the range of *Operator*. The *Operator* class represents all the possible operators that can be applied to ontology elements during the mappings. Fig. 2 shows the subclasses of the *Operator* class.

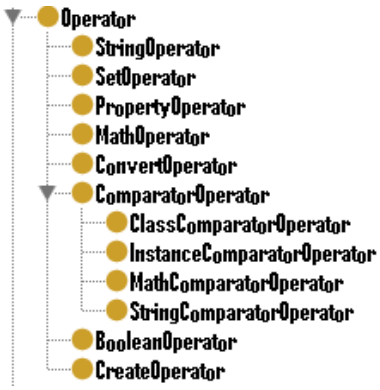

Fig. 2. Subclasses of the *Operator* class

Other than Boolean, mathematical and string operators, we have created the following operators to help with the mappings: 1. *SetOperator*: They are used to create intersection, unions and complements of classes. 2. *ConvertOperator*: Instances of the *ConvertOperator* that are convertToClass, convertToInstance and convertToProperty are used to convert any element of the source ontology to a class, instance or property respectively in the target ontology. 3. *CraeteOperator*: class is used for creating new elements in the target ontology during the mapping. 4. *ClassComparatorOperator*: Class comparators are used in the functions that compare classes to find sub-class, super-class and equivalence relations. 5. *InstanceOperator*: Instances of this class are equalInstance and notEqualInstance. The output of a function comparing two instances using equalInstnace is a Boolean variable with the value "true" if they are equivalent classes or with the value "false" otherwise. notEqualInstance works the opposite way.

D. Conditions

Mappings may be conditional. Property *hasCondition* with the domain of *Mapping* and range of *Condition* assigns conditions to a mapping. *Condition* class represents the conditions. *hasCardinalityType* with the domain of *Mapping* and the range *Cardinality* represents the cardinality type. Instance of the *Cardinality* are any, all, min and max. Data type property *hasNumericValueForCardinality* with the domain of *Mapping* shows the number of conditions that should be satisfied. A mapping whose condition satisfaction criterion is met is considered for mapping and instance transformation otherwise it is ignored. Using the abovementioned properties, one is able to express that at least three conditions of a mapping should be satisfied in order to participate in the mapping process.

V. Translation of MRO From OWL-Full to OWL-DL + SWRL

In order to use an instantiation of MRO (representing an ontology mapping) in the reasoning step of our semantic-based ontology mapping approach, we translate it to a combination of OWL-DL + SWRL or OWL2-RL + SWRL depending on the level of expressivity needed to represent the mapping. In this way, the translated mappings, the source and the target ontologies all can be regarded as a single ontology and an OWL reasoner can be used to improve the existing mappings by discovering new ones and perform instance transformation. Our translation algorithm performs the following steps on each mapping:

(1) Put all the non-output class variables in list1. Put all the output variables (Except for Boolean variables) in list2. Put all input Boolean output variables in list3.
(2) Translate the variables in list1 until no further transformation is possible.
(3) Translate the variables in list2 until no further transformation is possible.
(4) If list1 and list2 are empty, go to 5 else go to 2.
(5) Translate all the Boolean variables in list3 and process conditions.
(6) If all mappings are translated then go to 7 otherwise go to the next mapping
(7) Prepare the translated mapping for reasoning according to the translated variable.

Lists 1 and 2 are repeatedly swept for variables to be translated until both of the lists are empty. The reason is that translation of all of the output variables depends on the input variables and the translation of some of the input variables may depend on output variables. For example, an instance variable may belong to a class using property *classVariableHasClass* that is the output of a set function. In order to translate that instance variable, the class variable that it belongs to should be translated in list2 first. Steps 2, 3, 5 and 7 are further discussed in the following sub-sections.

A. Step 2 translation of list1

These variables are either translated to OWL constraints or SWRL axioms. If a variable has a value for one of the properties *hasInstanceVariable* or *classVariableHasValue*, it is translated to a SWRL axiom. In order to understand the translation process we go through the following example:

```
:cv1 a :ClassVariable;
 :hasInstanceVariable :personVar;
 :classVariableHasClass o1:Student;
 :classVariableHasProperty :hasWeight;
 :classVariableHasValue o1:weightVar.
```

Firstly, two SWRL variables are made with the name of the values of properties *hasInstanceVariable* and *classVariableHasValue* + "SWRLVar":

```
:personVarSWRLVar a swrl:Variable.
:weightVarSWRLVar a swrl:Variable.
```

Then a SWRL class atom is made to represent the class to which the created instance variable belongs. This class which is represented by the *classVariableHasClass* property is *o1:Student*:

```
[a swrl:ClassAtom ;
  swrl:argument1 :personVarSWRLVar;
  swrl:classPredicate o1:Student].
```

Finally, another axiom is created to show that the created SWRL variables are connected using the property indicated by the *classVariableHasProperty* that is *hasWeight* here:

```
[a swrl:DatavaluedPropertyAtom ;
   swrl:argument1 :personVarSWRLVar;
   swrl:argument2 :weightVarSWRLVar;
   swrl:propertyPredicate o1:hasWeight;]
```

Depending if the translated class variable belongs to the source or the target of the mapping, these created SWRL axioms are added to the body or the head of SWRL rule representing this mapping respectively.

If a variable is not translated to SWRL rules, it is translated to OWL axioms. Depending on the values of the properties *classPropertyQCROnClass*, *hasCardinalityType*, and *hasNumericValueForCardinality* a class variable is translated to a cardinality restriction in OWL-DL or a qualified cardinality restriction in OWL-2. In the following example, cv1 class variable represents instances that have maximum of two different values from the *SummerCource* class for the *hasCourse* property:

```
:cv1 a :ClassVariable
 :classPropertyQCROnClass o1:SummerCourse;
 :classVariableHasProperty o1:hasCourse;
 :hasCardinalityType :max;
 :hasNumericValueForCardinality "2"^^xsd:int.
```

The above example is translated to the following OWL triples:

```
[a owl:Restriction;
  owl:onClass o1:SummerCourse;
  owl:onProperty o1:hasCourse
  owl:maxQualifiedCardinality "2"^^xsd:int]
```

B. Step 3 translation of list2

Output variables with different operators are translated differently. For instance, set operators are translated to OWL axioms that make use of owl:intersectionOf, owl:unionOf etc. As an example, considering the following mapping function:

```
:func1 a :Function;
  :functionHasInputVariable1 o1:Male
  :functionHasInputVariable2 o1:Parent
  :functionHasOperator :intersectionSO
  :functionHasOutputVariable :func1OutVar.
```

This example is translated to:

```
:func1OutVar :variableHasClassValue
[a owl:class;
owl:intersectionOf( :Parent :Male)].
```

Output variables of functions that make use of mathematical operators are translated to SWRL rules that make use of SWRL built-ins. For instance, in order to add up two variables a and b and put the result in the variable c, we create the following function:

```
:a a :NumericVariable. :b a :NumericVarable.
:addFunc a : Function;
  :hasInput1 :a; :hasInput2 :b; :hasOutput :c;
  :hasOperator :mathDivide.
```

This example is translated to:

```
[a swrl:BuiltinAtom ;
 swrl:arguments (:outputSWRLVar :bSWRLVar
 :aSWRLVar); swrl:builtin swrlb:divide].
```

C. Step 5 translation of list3 and processing Conditions

Since OWL and SWRL do not support Boolean operators, mappings are first translated into a single mapping rule without considering the Boolean functions in it. Then we iterate through all the possible combination of values of the non-output Boolean variables and compute the values of the Boolean output variables in list3. As we iterate through the values, we create a copy of the existing SWRL rule created for the current mapping and add the SWRL axioms that represent the current values of both input and output Boolean variables. In this way, each rule is copied to several rules each representing a combination of the Boolean input variables. In this way, each created SWRL rule handles a specific combination of input Boolean variables.

In order to handle conditions, we go through the created rules in the previous step and discard the SWRL rules in which the assigned Boolean variables do not meet the condition satisfaction criteria. In this way, a great number of created SWRL rules are discarded in this step.

D. Step 7 preperation of the mappings for reasoning

Mappings represented by SWRL rules are ready for reasoning. However, relational mappings that are represented by OWL axioms need the final translation from OWL-Full to OWL-DL. During this translation, all the variables are replaced by their translated values. For instance, consider the following translated variable and mapping:

```
:func1OutVar :variableHasClassValue
```

```
 [a owl:class;
  owl:intersectionOf(o1:Parent o1:Male)].
:m1 a:RelationalMapping;
  :hasSource   o1:func1OutVar;
  :hasTarget   o2:Father;
  :hasRelation :subClassRelation
```

This example is translated to:

```
[a owl:class;
 owl:intersectionOf(o1:Parent o1:Male)
] rdfs:subClassOf o2:father.
```

VI. Evaluation

Mapping health informatics related ontologies especially CPG ontologies is usually a challenging task due to their high levels of expressivity. In order to evaluate the efficacy of our mapping representation language, we used it to map 3 CPG ontologies with a total of 9 instantiations to a general CPG representation ontology. During the mapping process, we did not come across a mapping pattern or an operator that was not supported by our mapping ontology. We translated the mappings to OWL + SWRL and performed reasoning on them in order to discover new mappings and to perform instance transformation. We executed all the 9 transformed instantiations using the execution engine developed in [16] for executing our general CPG ontology. We also executed these CPG in their original format using their own proprietary execution engine. We compared the execution results generated by our execution engine and the original execution engines for three imaginary patient scenarios. In all 9 cases, both execution engines generated the exact same recommendations. This indicates that the mapping has been accurate and the instances are transformed successfully. In all three mappings, the translation algorithm translated the mappings to either to OWL-DL or OWL 2-RL + SWRL. This is important to ensure the decidability of the process of discovering new mappings and instance transformation.

Comparison of our mapping ontology with the existing mapping representation languages against a comprehensive set of mapping patterns surveyed in [2] shows that our mapping representation ontology supports the widest range of these mapping patters. For instance, unlike most of these languages, our mapping ontology supports variables, meta-modelling and a wide range of operators that are needed for data manipulation and structural modifications. We also introduced the possibility of conditions and complex condition satisfaction criteria.

VII. Conclusion

In this paper, we introduced a new semantic-based ontology mapping approach based on semantic web technologies. We used our approach to map three CPG ontologies to a general CPG ontology and to transform their instances. Execution results showed that our approach represents the mapping accurately and performs instance transformation correctly. Our mapping approach has three advantages over existing mapping approaches: (1) higher levels of expressivity; (2) better shareability and acceptance due to support by several semantic web tools developed for manipulation, visualization and reasoning; (3) Formal semantics in OWL and SWRL that enables us to improve the existing mappings and perform instance transformation automatically in a semantic-based ontology mapping approach. For future work, we are interested in using the functions provided by either SQWRL or SPARQLE query languages to improve the mapping representation expressivity.

Acknowledgements: This research project is sponsored by a research grant from Green Shield, Canada. The authors acknowledge the support of Green Shield, Canada.

References

[1] J. Euzenat. "An API for ontology alignment," in The Proceeding Of Semantic Web ISWC 2004, S. McIlraith, D. Plexousakis and F. Harmelen, Eds. Berlin Heidelberg: Springer, 2004, pp. 698-712.

[2] F. Scharffe, J. Bruijn, D. Foxvog. "D 4.3.2 ontology mediation patterns Library V2," Deliverable D4.3.2, EU-IST Integrated Project (IP) IST-2003-506826 SEKT, 2006.

[3] F. Scharffe, A. Zimmermann, "D 2.2.10 Expressive alignmentlanguage and implementation", Deliverable D2.2.10 EU-IST Integrated Project (IP) IST-2004-507482 SEKT, 2007.

[4] A. Maedche, B. Motik, N. Silva and R. Volz, "MAFRA - A MApping FRAmework for distributed ontologies," in Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, 2002, pp. 235-250.

[5] A. Sotnykova, C. Vangenot, N. Cullot, N. Bennacer and M. Aufaure, "Semantic mappings in description logics for spatio-teMROral database schema integration," in Journal on Data Semantics III, S. Spaccapietra and E. Zimányi, Eds. Berlin / Heidelberg: Springer, 2005, pp. 586-586.

[6] C. Meilicke, H. Stuckenschmidt and A. Tamilin, "Improving automatically created mappings using logical reasoning." in Ontology Mapping Workshop at ISWC, Athens, GA, USA, 2006, pp. 61-72.

[7] D. Calvanese, G. D. Giacomo and M. Lenzerini, "Ontology of integration and integration of ontologies," Description Logics, vol. 49, pp. 10-19, 2001.

[8] H. Stuckenschmidt, L. Serafini and H. Wache, "Reasoning about ontology mappings," ITC-IRST, Trento, 2005.

[9] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini and H. Stuckenschmidt, "C-OWL: Contextualizing ontologies." in International Semantic Web Conference, 2003, pp. 164-179.

[10] J. Euzenat and P. Shvaiko, Ontology Matching. Springer-Verlag: New York Inc, 2007.

[11] E. Goldberg and Y. Novikov, "BerkMin: A fast and robust SAT-solver," in Proceedings of Design, Automation and Test in Europe Conference and Exhibition. 2002, pp. 142-149.

[12] J. P. Marques-Silva and K. A. Sakallah, "GRASP: a search algorithm for propositional satisfiability," IEEE Trans. Comput., vol. 48, pp. 506-521, 1999.

[13] H. Thomas, D. Sullivan and R. Brennan, "Ontology Mapping Representations: a Pragmatic Evaluation," Management, pp. 228-232, 2009.

[14] N. F. Noy and M. A. Musen, "Anchor-prompt: Using non-local context for semantic matching," in Proceedings of the Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001), 2001, pp. 63-70.

[15] J. d. Bruijn and A. Polleres, "Towards an ontology mapping specification language for the semantic web," DERI - DIGITAL ENTERPRISE RESEARCH INSTITUTE, Tech. Rep. DERI-2004-06-30, 2004.

[16] B. Jafarpour, S. Abidi and S. Abidi. "Exploiting OWL reasoning services to execute ontologically-modeled clinical practice guidelines," in Proceedings of the 13th conference on Artificial intelligence in Medicine, M. Peleg, N. LavraÄ and C. Combi, Eds. Berlin Heidelberg: Springer-Verlag, 2011, pp. 307-311.

[17] B. Motik, U. Sattler and R. Studer, "Query Answering for OWL-DL with rules," Web Semant. Sci. Serv. Agents World Wide Web, vol. 3, pp. 41-60, 2005.