Applying Reasoning to Instance Transformation

Adrian Mocan, Mick Kerrigan and Emilia Cimpian

Semantic Technology Institute Innsbruck
Leopold-Franzens Universität Innsbruck, Austria
firstname.lastname@sti2.at

Abstract. Significant effort has been invested by the Semantic Web community in methodologies and tools for creating ontology mappings. Various techniques for mapping creation have been developed; however less interest has been shown in the actual usage of the created mappings and their application in concrete mediation scenarios. In this paper we show how mappings can be converted to logical rules and evaluated by a Datalog reasoner in order to produced the desired mediation result. The mediation scenario discussed in this work is *instance transformation*: data expressed as ontology instances is transformed from the terms of one ontology in to the terms of another ontology based on mappings created in a design-time phase. We analyze the required reasoning task and describe how the mappings created at design-time can be grounded at runtime. We also explore strategies and techniques that we employed in order to improve the efficiency of the overall process.

1 Introduction

Data mediation in the context of ontologies involves the creation of mappings between ontologies at the schema level. The set of mappings (or *alignment*) is created in advance between two or more ontologies so they can be used automatically at run-time to solve various heterogeneity problems in a given mediation scenario. While the process of creating ontology alignments has been well explored in the last decade in the Semantic Web community [4,14,11], the usage of the alignments has so far been considered an application-specific detail.

This paper describes the usage of ontology mappings in the instance transformation scenario. It shows how a relatively simple mapping representation format can be grounded automatically to complex logical rules. After the grounding is applied, a general-purpose Datalog-based reasoner can be used to evaluate the rules and to retrieve the mediated data by query. We argue that by separating the mappings representation from their executable form offers advantages that can be exploited in applying a set of optimizations on the overall instance transformation process. We propose a grounding mechanism capable of transforming mappings from an abstract form into executable mapping rules and go on and provide an overview of several optimization techniques suited for this scenario.

Section 2 given an overview of the main technologies used in our approach, while Section 3 describes the grounding of abstract mappings to mapping rules

and the main reasoning task that is being performed. Section 4 briefly introduced several optimization types that can be applied in this context, and we close the paper with a presentation of related work and some conclusions.

2 Ontology Mappings

The work presented in this paper is part of a broader context, namely the Web Service Execution Environment (WSMX) [10], and it targets ontologies that conform to the Web Service Modeling Ontology (WSMO) [13]. WSMO ontologies are expressed using the Web Service Modeling Language (WSML) [3], a language which offers several language variants. For the approach used in this paper we consider the Logic Programming branch of WSML, namely the WSML-Flight and WSML-Rule [3] variants. **WSML-Flight** is a powerful rule language based on a logic programming variant of F-Logic [8] and is semantically equivalent to Datalog with inequality and (locally) stratified negation. **WSML-Rule** extends WSML-Flight with further features from Logic Programming, namely the use of function symbols, unsafe rules, and unstratified negation under the well-founded semantics.

The instance transformation scenario relies on a set of pre-existing mappings (i.e. an alignment) between the source and the target ontologies. This paper does not discuss the methods used in creating the alignments between the ontologies - they could be either entirely manual or semi-automatic assisted by graphical tool like the one we proposed in [9]. In this scenario we choose not to directly represent the ontology mappings using a specific WSML variants, but instead to use an intermediary, abstract representation. The Abstract Mapping Language (AML), a language proposed in [2] (extended and elaborated in [5] as the *Alignment Format*) is used to express the mappings, because it does not commit to any existing ontology representation language. The abstract mappings state that a semantic relationship exists between the mapped entities, but the actual semantics and interpretation is associated in a second step, called *grounding* (see Section 3.1). There are a number of reasons behind this design decision:

Reusability: The same set of mappings can be used in various mediation scenarios. A grounding mechanism can be applied at runtime and a formal semantics can be associated with these mappings in order to suit the targeted scenario.

Manageability: As ontologies evolve, the set of mappings between them must be updated. If mappings are represented as rules in a particular ontology language the special features and peculiarities of this language are reflected in the rules as well.

Although the AML has its own syntax, in this paper we will use a more schematic representation of the mappings in examples, both for brevity and for emphasizing the simplicity of this form of mapping representation. As such, for every mapping we use the following form:

$$mapping(\ sourceEntity, [sourceEntityRestriction],\\ targetEntity, [targetEntityRestriction], typeOfMaping)$$

The source and target entity restriction are optional conditions on the mapped entities, and $typeOfMapping \in \{\mathcal{C}2\mathcal{C}, \mathcal{C}2\mathcal{A}, \mathcal{A}2\mathcal{C}, \mathcal{A}2\mathcal{A}\}$ where $\mathcal{C}2\mathcal{C}$ stands for a concept to concept (or class to class) mapping, $\mathcal{A}2\mathcal{A}$ stands for an attribute to attribute mapping[1], $\mathcal{A}2\mathcal{C}$ stands for a attribute to concept mapping, and $\mathcal{C}2\mathcal{A}$ stands for a concept to attribute mapping. The AML can be also used to express mappings between relations of arbitrary arity, but here due to space constraints we restrict ourselves to mappings involving only concepts and attributes.

3 Instance Transformation

The instance transformation scenario can be summarized as follows: different business actors use ontologies to describe their internal business logic and their data. Each of these actors use their own information system and they interact with each other as part of arbitrary business processes. However as the ontologies of each of the actors is likely to be different there is a need for a specialized service capable of transforming data expressed in terms of a given ontology (the source ontology) into the terms of another ontology (the target ontology), allowing the two actors to communicate effectively, without changing the way they represent their data. As the instance transformation occurs at run-time it has to be performed completely automatically using mappings that have already been created at the schema level during a design-time phase.

3.1 Grounding

Each of the abstract mappings have to be grounded to rules expressed in WSML in order for them to be used in a reasoner. WSML-Rule has been preferred over WSML-Flight since it offers function symbols, which allows the building and usage of constructed terms as new identifiers for the mediated instances based on the identifiers of the original source instances. This identifiers of the target instances are built using the function symbol *mediated* with two parameters: first is the source instance out of which the target instance is derived and second is the target concept the source instance is mediated to (e.g. $mediated(johnS, o2\#Citizen)$).

Formula 1 shows the grounding of a concept to concept mapping ($\mathcal{C}2\mathcal{C}$) to WSML, where Γ is a grounding function that takes as parameter an abstract mapping (or a fragment of a mapping, e.g. restrictions) and produces a WSML rule[2].

$$
\begin{aligned}
&\Gamma(mapping\ (C_S, C_S^R, C_T, C_T^R, \mathcal{C}2\mathcal{C})) \mapsto \\
&\quad mediated(?x, C_T)\ \textbf{memberOf}\ C_T\ \textbf{and}\ mapped(C_S, C_T, ?x) :- \\
&\quad\quad ?x\ \textbf{memberOf}\ C_S\ \textbf{and}\ \Gamma(C_S^R)\ \textbf{and}\ \Gamma(C_T^R).
\end{aligned} \tag{1}
$$

[1] In WSML attributes are binary relations local to the concept they are defined on and they can have both a data-type or another concept as range.

[2] In WSML variables are preceded by a "?", class memberships are denoted by "**memberOf**", conjunctions by "**and**" and inheritance relationships by "**subConceptOf**". Also, $\alpha[\beta$ **hasValue** $\gamma]$ and $\alpha[\beta$ **ofType** $\gamma]$ are atomic formulas called molecules; both α and γ identify instances (or values) and concepts (or data-types) respectively, while β identifies an attribute.

Formula 2 describes the grounding of the $\mathcal{A}2\mathcal{A}$ mappings when both the source and the target attributes have as range a data-type value. In this case the source value is simply copied to the target instance. Lines 3 and 4 assure that this rule covers also the cases when the mapped source attribute is inherited from an upper concept and the actual instance to be mediated is an instance of a specialized concept (denoted by the $?subC_S$ variable).

$$
\begin{aligned}
1 \quad & \Gamma(mapping\ (C_S.A_S, A_S^R, C_T.A_T, A_T^R, \mathcal{A}2\mathcal{A})) \mapsto \\
2 \quad & \quad mediated(?x, C_T)[A_T\ \textbf{hasValue}\ ?v]\ \textbf{memberOf}\ C_T :- \\
3 \quad & \quad\quad ?x[A_S\ \textbf{hasValue}\ ?v]\ \textbf{memberOf}\ ?subC_S\ \textbf{and} \\
4 \quad & \quad\quad ?subC_S\ \textbf{subConceptOf}\ C_S\ \textbf{and}\ mapped(?subC_S, C_T, ?x)\ \textbf{and} \\
5 \quad & \quad\quad \Gamma(A_S^R)\ \textbf{and}\ \Gamma(A_T^R). \qquad (2)
\end{aligned}
$$

In Formula 3 the mappings between attributes having as ranges other concepts are addressed. As shown at line 2, the value of the target attribute is set to another mediated instance, which is produced by one of the rules generated by Formulas 2 to 5. At line 5, it is checked if this other mediated instance has any attributes, in order to avoid the generation of meaningless instances due to incomplete mappings.

$$
\begin{aligned}
1 \quad & \Gamma(mapping\ (C_S.A_S, A_S^R, C_T.A_T, A_T^R, \mathcal{A}2\mathcal{A})) \mapsto \\
2 \quad & \quad mediated(?x, C_T)[A_T\ \textbf{hasValue}\ mediated(?i, SubR_T)]\ \textbf{memberOf}\ C_T :- \\
3 \quad & \quad\quad ?x[A_S\ \textbf{hasValue}\ ?i]\ \textbf{memberOf}\ ?subC_S\ \textbf{and} \\
4 \quad & \quad\quad ?subC_S\ \textbf{subConceptOf}\ C_S\ \textbf{and}\ mapped(?subC_S, C_T, ?x)\ \textbf{and} \\
5 \quad & \quad\quad mediated(?i, SubR_T)[?anAttribute\ \textbf{hasValue}\ ?aValue]\ \textbf{and} \\
6 \quad & \quad\quad \Gamma(A_S^R)\ \textbf{and}\ \Gamma(A_T^R). \qquad (3)
\end{aligned}
$$

The concept $SubR_T$ is extracted during the grounding process based on the range of the mapped target attribute, i.e. a distinct rule is generated for the attribute's range and for each of the range's subclasses (since every instance of the attribute's range sub-concept is a valid filler of that attribute). As described in [9], no mappings between two attributes having as range a data-type and a concept, are allowed (compensated by the usage of $\mathcal{C}2\mathcal{A}$ and $\mathcal{A}2\mathcal{C}$ mappings).

The grounding of the $\mathcal{A}2\mathcal{C}$ mappings , as seen in Formula 4, handles the cases when different levels of aggregation are used in the ontologies, e.g. the attribute $C_S.A$ has the concept CR_S as range, and CR_S's instances also need to be transformed into instances of the C_T. This type of mapping can be applied only when CR_S is a concept, but not a data-type.

$$
\begin{aligned}
1 \quad & \Gamma(mapping\ (C_S.A_S, A_S^R, C_T, C_T^R, \mathcal{A}2\mathcal{C})) \mapsto \\
2 \quad & \quad mediated(?x, C_T)[?anA\ \textbf{hasValue}\ ?aValue]\ \textbf{memberOf}\ C_T :- \\
3 \quad & \quad\quad ?x[A_S\ \textbf{hasValue}\ ?i]\ \textbf{memberOf}\ C_S\ \textbf{and} \\
4 \quad & \quad\quad mediated(?i, C_T)[?anA\ \textbf{hasValue}\ ?aValue]\ \textbf{memberOf}\ C_T\ \textbf{and} \\
5 \quad & \quad\quad \Gamma(A_S^R)\ \textbf{and}\ \Gamma(C_T^R). \qquad (4)
\end{aligned}
$$

The Formula 5 is the symmetric of Formula 4, grounding the $\mathcal{C}2\mathcal{A}$ mappings.

$$
\begin{aligned}
1 \quad & \Gamma(mapping\ (C_S, C_S^R, C_T.A_T, A_T^R, \mathcal{C}2\mathcal{A})) \mapsto \\
2 \quad & \quad mediated(?x, C_T)[A_T\ \textbf{hasValue}\ mediated(?x, SubR_T)]\ \textbf{memberOf}\ C_T :- \\
3 \quad & \quad\quad ?x\ \textbf{memberOf}\ C_S\ \textbf{and} \\
4 \quad & \quad\quad mediated(?x, SubR_T)[?anA\ \textbf{hasValue}\ ?aValue]\ \textbf{memberOf}\ SubR_T\ \textbf{and} \\
5 \quad & \quad\quad \Gamma(C_S^R)\ \textbf{and}\ \Gamma(A_T^R). \qquad (5)
\end{aligned}
$$

The restrictions can be seen as conditions that have to hold in order for the mapping to apply. They are divided into three main classes: *attribute occurrence (aoc)*, *attribute type (atc)* and *attribute value (avc)* conditions. Due to space constraints, this paper does not include a complete and detailed description of the conditions grounding. However, the Formulas 6 to 9 shows the grounding of the conditions when the "equals" operator is used. The grounding in Formulas 6 and 7 can be applied on any conditions, while, in order to avoid the creation of unsafe rules, Formula 8 is used to ground the only the *avc* set on source entities, while Formula 9 is used to ground only the *avc* set on the target entities.

$$\Gamma(atc(C.A, "equals", R)) \mapsto C[A \textbf{ ofType } R] \tag{6}$$

$$\Gamma(aoc(C.A)) \mapsto C[A \textbf{ ofType } ?aRange] \tag{7}$$

$$\Gamma(avc(C.A, "equals", Value)) \mapsto C[A \textbf{ hasValue } Value] \tag{8}$$

$$\begin{aligned}\Gamma(avc(C.A, "equals", Value, tVariable)) \mapsto \\ assignement(Id, ?tVariable) \textbf{ and } C[A \textbf{ hasValue } ?tVariable] \\ assignement(Id, Value).\end{aligned} \tag{9}$$

The Id uniquely identifies different assignments, one distinct fact is generated each time a value from an target *avc* has to be assigned to a target attribute.

3.2 Reasoning Task

After grounding, the alignment can be seen as an ontology that imports the source and the target ontologies and the set of mapping rules.

Table 1. Source and a target ontology fragments (expressed in WSML [3])

Source Ontology (o1)

```
ontology BelgianCitizenOntology
  concept Person
    hasChristianName ofType (0 *) _string
    hasSurname ofType (1 1) _string
    hasGender ofType (1 1) Gender
    dateOfBirth ofType Date

  concept Date
    day ofType _integer
    month ofType _integer
    year ofType _integer

  concept Gender
  instance _1 memberOf Gender
  instance _2 memberOf Gender
  instance _ memberOf Gender
```

Target Ontology (o2)

```
ontology ItalianCitizenOntology
  concept Citizen
    hasName ofType (1 1) Name
    hasSex ofType (1 1) Sex
    hasBirthday ofType (1 1) Date

  concept Name
    hasFirstName ofType (1 1) _string
    hasSurname ofType (1 1) _string

  concept Date
    day ofType _integer
    month ofType _integer
    year ofType _integer

  concept Sex
  instance M memberOf Sex
  instance F memberOf Sex
  instance N memberOf Sex
```

Conceptually, it could be also seen as a "merged ontology" where the input ontologies were independently put together (the separation is realized by namespaces) and linked by rules. By adding the source data to it and posing queries in terms of the target ontology, the appropriate mapping rules are triggered and the mediated data produced as a result.

Considering the two ontology fragments in Table 1, a set of mappings as the one presented in Listing 1.1 can be created.

Listing 1.1. Abstract mappings

```
mapping(o1#Person, o2#Citizen, C2C) mapping(o1#Person,
o2#Citizen.hasName, C2A) mapping(o1#Person, o2#Name,
C2C) mapping(o1#Person.hasChristianName,
o2#Name.hasFirstName, A2A) mapping(o1#Person.hasSurname,
o2#Name.hasSurname, A2A) mapping(o1#Person.hasGender,
avc(o1#Person.hasGender, equals, o1#_1),
        o2#Citizen.hasSex, avc(o2#Citizen.hasSex, equals, o2#M), A2A)
mapping(o1#Person.hasGender, avc(o1#Person.hasGender, equals, o1#_2),
        o2#Citizen.hasSex, avc(o2#Citizen.hasSex, equals, o2#F), A2A)
mapping(o1#Person.hasGender, avc(o1#Person.hasGender, equals, o1#_),
        o2#Citizen.hasSex, avc(o2#Citizen.hasSex, equals, o2#N), A2A)
mapping(o1#Gender, o2#Sex, C2C)
mapping(o1#Person.dateOfBirth, o2#Citizen.hasBirthday, A2A)
mapping(o1#Date, o2#Date, C2C)
mapping(o1#Date.day, o2#Date.day, A2A)
mapping(o1#Date.month, o2#Date.month, A2A)
mapping(o1#Date.year, o2#Date.year, A2A)
```

Listing 1.2 shows the mapping rules generated by grounding the abstract mappings depicted in Listing 1.1, using the Formulas 1 to 9. These mapping rules are expressed as WSML axioms and they have a precise semantics, as defined by WSML-Rule [3]. They are included in an ontology that imports the source and the target ontologies. The source instances to be mediated are also included in this ontology, which is then registered in the reasoner. Listing 1.3 depicts a set of source instances to be mediated by using the mappings shown above.

Listing 1.2. WSML mapping rules between the Belgian and Italian ontologies

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace {o1 _"http://www.semantic-gov.org/BelgianCitizenOntology#",
           o2 _"http://www.semantic-gov.org/ItalianCitizenOntology#"}

ontology merged_ontology
importsOntology { o1#BelgianCitizenOntology, o2#ItalianCitizenOntology}

axiom o2#ccMappingRule18 definedBy
    o2#mappedConcepts(o1#Person,o2#Citizen,?X17) and
    o2#mediated(?X17,o2#Citizen) memberOf o2#Citizen :- ?X17 memberOf o1#Person.

axiom o2#caMappingRule72 definedBy
    o2#mediated(?X69,o2#Citizen)[o2#hasName hasValue
        o2#mediated1(?X69,o2#Name)] memberOf o2#Citizen :- ?X69 memberOf o1#Person
            and o2#mediated1(?X69,o2#Name)[?A70 hasValue ?V71] memberOf o2#Name.

axiom o2#ccMappingRule12 definedBy
    o2#mappedConcepts(o1#Person,o2#Name,?X11) and
    o2#mediated(?X11,o2#Name) memberOf o2#Name :- ?X11 memberOf o1#Person.

axiom o2#aaMappingRule48 definedBy
    o2#mediated(?X45,o2#Name)[o2#hasFirstName hasValue ?Y46] memberOf o2#Name :-
        ?X45[o1#hasChristianName hasValue ?Y46] memberOf ?SC47 and
        ?SC47 subConceptOf o1#Person and o2#mappedConcepts(?SC47,o2#Name,?X45).
...
axiom o2#aaMappingRule84 definedBy
    o2#mediated(?X79,o2#Citizen)[o2#hasSex hasValue ?Y81] memberOf o2#Citizen :-
     ?X79[o1#hasGender hasValue ?Y80] memberOf ?SC83 and
     ?SC83 subConceptOf o1#Person and o2#mappedConcepts(?SC83,o2#Citizen,?X79)
     and ?X79[o1#hasGender hasValue o1#_1] and o2#assignement_82(?Y81).

axiom o2#assignement_82 definedBy
    o2#assignement_82(o2#M).
...
axiom o2#ccMappingRule14 definedBy
    o2#mappedConcepts(o1#Gender,o2#Sex,?X13) and
    o2#mediated(?X13,o2#Sex) memberOf o2#Sex :- ?X13 memberOf o1#Gender.
```

Once this ontology is registered in the reasoner, queries can be asked to retrieve the mediated data. The mediation take place within the reasoner when the rules are evaluated and the target instances become implicit knowledge in

the reasoning space. Normally, one or more instances from the source set are considered to be root instances and used in determining a starting query point. Otherwise, the procedure exemplified below must be applied to all the instances in the source set (although this could lead to redundant computations).

Listing 1.3. Source instances to be mediated

```
namespace {o1 -"http://www.semantic-gov.org/BelgianCitizenOntology#",
           o2 -"http://www.semantic-gov.org/ItalianCitizenOntology#"}

instance johnS memberOf o1#Person
    o1#hasSurname hasValue "Smith"
    o1#hasChristianName hasValue "John"
    o1#hasGender hasValue o1#_1
    o1#dateOfBirth hasValue johnS_birthDate

instance johnS_birthDate memberOf o1#Date
    o1#day hasValue 11
    o1#month hasValue 4
    o1#year hasValue 1980
```

Assuming that *johnS* is the root instance in the set shown in Listing 1.3, the first step is to identify which is the most relevant concept from the target to be mediated to. *johnS* is an instance of concept *Person* and there are two concepts in the target the *Person* is mapped to: *Citizen* and *Name*. Since the concept *Name* can be reached from *Citizen* via the *hasName* attribute (paths of any length can be explored in this way), the concept *Citizen* is considered as being an *ancestor* of the concept *Name* and by this, more suited for the instance *johnS* to be mapped to[3].

Once the concept *Citizen* is selected, queries can be posed to the reasoner. The first query and the results obtained are shown in Listing 1.4.

Listing 1.4. The target instance obtained by mediating the *johnS* instance

```
?- ?x memberOf o2#Citizen.

Found < 1 > results to the query:
(1) -- ?x = mediated(johnS,o2#Citizen)
```

The target instance *mediated(johnS,o2#Citizen)* has to be explored and its attributes and their values retrieved. The next query and the obtained answers are shown in Listing 1.5. For each of the attributes (*?y*) this query retrieves both the value (*?z*) and the type of this value (*?avC*).

Listing 1.5. Finding the attributes and their values for a target instance

```
?-mediated(johnS,o2#Citizen)[?y hasValue ?z] memberOf o2#Citizen and ?z memberOf ?avC.
Found < 3 > results to the query:
(1) -- ?z = o2#M, ?avC = o2#Sex,?y =o2#hasSex
(2) -- ?z = mediated(johnS,o2#Name), ?avC = o2#Name, ?y = o2#hasName
(3) -- ?z = mediated(johnS_birthDate,o2#Date), ?avC = o2#Date, ?y = o2#hasBirthday

?-mediated(johnS,o2#Name)[?y hasValue ?z] memberOf o2#Name and ?z memberOf ?avC.
Found < 2 > results to the query:
(1) -- ?z = John, ?avC = _string, ?y = o2#hasFirstName
(2) -- ?z = Smith, ?avC = _string, ?y = o2#hasSurname
```

[3] In the presence of an inheritance hierarchy, if for example there are multiple mappings from the concept *Person* to several concepts in this hierarchy, the most specific concept would qualify.

For each of the attribute values having as type a concept the query process continues recursively. Listing 1.5 illustrates the queries for the *hasName* attribute and *mediated(johnS,o2#Name)* value.

By applying this querying mechanism all the target instances can be retrieved and materialized. The obtained mediated instances are shown in Listing 1.6.

Listing 1.6. Mediated target instances

```
instance mediated(johnS, o2#Citizen) memberOf o2#Citizen
  o2#hasSex hasValue o2#M
  o2#hasName hasValue o2#mediated(johnS, o2#Name)
  o2#hasBirthday hasValue mediated(johnS_birthDate, o2#Date)
instance mediated(johnS, o2#Name) memberOf o2#Name
  o2#hasFirstName hasValue "John"
  o2#hasSurname hasValue "Smith"
instance mediated(johnS_birthDate, o2#Date) memberOf o2#Date
  o2#day hasValue 11
  o2#year hasValue 1980
  o2#month hasValue 4
```

The logic program corresponding to the "merged ontology" is decidable even if this ontology is expressed in WSML-Rule, which is generally undecidable [3]. The reason is that the function symbols are used in such a way that only one level of constructed terms can be generated. That is, the mapping rules will never generate terms like $mediated(mediated(...(X, C)...))$. Formulas 1 to 5 builds the constructed term $mediated(?x, C_T)$ **memberOf** C_T only if $?x$ **memberOf** C_S exists. Since C_S and C_T are concepts from the source and the target ontology, respectively, they are separated by namespaces and, as a consequence, distinct.

4 Effective Data Mediation

In this section we introduce the lessons learned from applying the instance transformation scenario described in Section 3 to use cases in two EU funded projects, namely SEEMP and SemanticGov[4]. SEEMP enables the exchange of data between different Employment Services in Europe that use their own data structures and taxonomies. SemanticGov aims to build the infrastructure necessary to enable the provision of Semantic Web Services for Public Administration.

4.1 Optimizations for the Instance Transformation Scenario

In the context of the SEEMP project, mappings between the local ontology of each employment service and the reference ontology of the marketplace were made at design time using the mapping tools [9] in the Web Service Modeling Toolkit (WSMT) [7], thus transforming an instance from one local employment service to another was done by transforming the source instance into terms of the reference ontology and then transforming this instances into terms of the other local ontology. However having created the mappings it was obvious that the performance of the reasoning when performing instance transformation was an issue. With further investigation it became apparent that the number of facts and rules registered in the reasoner was very high. Essentially for a given instance transformation the size of the source and target ontologies was very large

[4] For more details see `http://www.seemp.org` and `http://www.semantic-gov.org`.

and the mappings between these two large structures was causing the forward chaining algorithms within the reasoner to require significant computation time. Also the instance data to transform was very large and parts of this information where untransformable due to a lack of mappings. In this section we explore the optimizations performed in the SEEMP project to improve the performance of the overall instance transformation process.

Source Instance Filtering: In a given scenario a certain amount of the input source instances can be transformed based on the coverage of the available mappings. In scenarios where this coverage is low the reasoner contains lots of instance data that will never be transformed to the target ontology. This additional information is unnecessary for the instance transformation step, but having it within the reasoner means that it gets used in the model computations and thus its removal would improve the overall performance. To this end a source instance pre-filter was added to the instance transformation process that removes those parts of the source instances that cannot be transformed. This filter can have quite dramatic effects on the quantity of data loaded into the reasoner as the absence of one attribute to attribute mapping can remove entire branches of source instances. Having implemented this filter an improvement of between 5% and 10% in most of our test cases was observed. The amount of performance improvement that can be gained by applying this filter is proportional to the amount of extra source data that is untransformable. Thus in scenarios where there is 100% coverage the filter will not remove any content and the performance will remain the same.

Mapping Filtering: In other cases the source instance set can be very small and the number of mappings very large. The extra mappings that are being loaded into the reasoner will only partially fire without producing any meaningful and valid result. Further exploration into the SEEMP project test cases showed that this was not an extraordinary scenario but in fact the normal case. Thus a mapping filter was added to the instance transformation process removing those mappings unrelated to the source instances. This filter showed a huge performance improvement of between 40% and 50% for the average test case in the SEEMP project as the ontologies in the test cases contain a number of very large taxonomies with many mappings between them, approximately 70% of the mappings created at design time are filtered away at runtime by the mapping filter for these test cases.

Source and Target Ontology Filtering: Having filtered both the source instances and the mappings the bulk of the data remaining in the reasoner is the source and target ontologies. Dynamically filtering ontologies prior to putting them in a reasoner is a non trivial affair; however initial research appears to show that a naive filter of the ontologies based on the source instances and the available mappings can be used to remove those parts of the source and target ontologies which are unnecessary for a given instance transformation process. Our ongoing research seems to show that a 50% to 60% performance improvement on top of the improvements already garnered by the source instance and mapping filters can be gained.

The combination of these filters can dramatically reduce the amount of data that is loaded in the reasoner at runtime and therefore the performance of the overall instance transformation process. Crucially these filters are all dynamic and applied at runtime thus they involve no human effort in order to configure them. In our current system each of these filter is turned on and off via a flag, thus they can be applied in those scenarios where they are desirable and turned off where they are not.

4.2 External Value Transformation Services

There are situations when the data-values embedded in the source instances need to be transformed before they can be assigned to the attributes part of the target instances. Such transformations could involve simple string concatenations or more complex, dynamic transformations based on external factors, for example currency conversions.

Listing 1.7. Example of a value transformation service usage

```
axiom o2#aaMappingRule definedBy
  o2#mediated(?X45,o2#Citizen)[o2#hasName hasValue
    http://example.org/concatService(?Y46, ?Y47)] memberOf o2#Citizen:-
      ?X45[o1#hasChristianName hasValue ?Y46] memberOf o1#Person and
      ?X45[o1#hasSurname hasValue ?Y47] memberOf ?SC47 and
      ?SC47 subConceptOf o1#Person and o2#mappedConcepts(?SC47,o2#Citizen,?X45).

instance mediated(johnS, o2#Citizen) memberOf o2#Citizen
  o2#hasSex hasValue o2#M
  o2#hasName hasValue http://example.org/concatService("John", "Smith")
  o2#hasBirthday hasValue mediated(johnS_birthDate, o2#Date)
```

Normally, this kind of conversions requires either the use of specialized data manipulation functions within the reasoning environment or the access to external services that can apply the conversion after the reasoning occurred. Since the first option would significantly limit the type of the allowed transformations, we rely on the second approach. The domain experts can specify during the creation of mappings, an arbitrary identifier to denote the service they would like to use. At run-time, the reasoner is used only to assign to the target recipient a string encoding of the service and its parameter. After the mediated target instances are retrieved from the reasoner, a post-processing phase identifies and evaluates every service/parameter encoding.

For example, assuming that the attribute *hasName* of the *Citizen* concept in that target ontology would have as range a *string*, representing a concatenation of the first name and the surname of a person, such a value transformation service would be required. Listing 1.7 shows the corresponding WSML mapping rule and the attribute value produced after reasoning.

The services set that can be used in the mapping process should be customizable at the mapping creation tool level; they could be added, removed or briefly described. It is important to note that at that level no implementation has to be provided - the implementation has to be available and integrated with the instance transformation component only at run-time.

5 Related Work

TSIMMIS [6] is a system for integrating information coming from heterogeneous data sources. TSIMMIS includes a mediator-generator able to produce mediator descriptions in the Mediator Specification Language (MSL) [12]. MSL and the abstract mapping language's grounding (AMLg) proposed in this paper are based on the same general principles: they both rely on datalog-like rules which construct target data, based on the information from the sources. However MSL relies on patterns to match and create data from the source and target data sets, while AMLg acts on schema level elements. Additionally, using MSL, one needs to construct paths from root objects down to the relevant information in the model, while with AMLg separate rules are created. These rules are eventually "composed" by a reasoner in order to produce the desired target data out of the source instances. This strategy assures that when new schema elements need to be included in the mappings set, no re-engineering of the existing mappings or rules is necessary. Another relevant similarity aspect between TSIMMIS and the approach in this paper, is the usage of Skolem functions [8] or function symbols. Both approaches use them to encode information regarding how the target object or instance has been derived from the source. While this information is used by MSL only to build special target objects, it plays a crucial role for AMLg allowing the combination of the results from individual rules into one complex result instance.

Abiteboul and colleagues [1] propose a middleware data model, as basis for the integration task, and declarative rules to specify the integration. The model is a minimal one and the data structure consists of ordered label trees. The authors also consider two types of rules: *correspondence rules* used to express relationships between the nodes of their model (similar with our mappings in the AML but expressed in Datalog) and *translation rules*, a decidable sub-case for the actual data translation (resembling our mapping rules expressed in WSML).

6 Conclusions

This paper describes an approach that uses reasoning to perform data mediation, based on pre-existing alignments between ontologies. The alignments consists of mappings expressed in the Abstract Mapping Language, a form of representation that does not commit to any ontology representation language or formalism. In order to apply these mappings in a concrete mediation scenario they have to be grounded to a concrete language and to have a formal semantics associated. This paper provides a grounding of the AML to WSML-Rule, suited for the *instance transformation* scenario. A detailed example of the reasoning task is also provided together with a set of optimization that can be applied in order to improve the performances of the instance transformation process. Additionally, the paper describes a way of using value transformation services in conjunction with reasoning without being restricted to the reasoner's set of built-ins.

A full implementation of the instance transformation component is available as part of WSMX, and it is available for download at `http://sourceforge.net/projects/wsmx/`. As future work we plan to conduct a full evaluation of the performance improvements discussed in this paper using the showcases developed in EU funded projects where this prototype is being developed and used.

7 Acknowledgements

The work is funded by the European Commission under the projects KnowledgeWeb, SEEMP, SemanticGov, SUPER and SHAPE; by the FFG (Österreichische ForschungsFörderungsGesellschaft mbH) under the project SemBiz.

References

1. S. Abiteboul, S. Cluet, and T. Milo. Correspondence and Translation for Heterogeneous Data. In *Proc. of the 6th Intl Conf on Database Theory (ICDT-1997)*, pages 351–363, Delphi, Greece, 1997. Springer-Verlag.
2. J. de Bruijn, D. Foxvog, and K. Zimmerman. Ontology Mediation Patterns Library. SEKT Project D4.3.1, available at: `http://www.sekt-project.com`, 2004.
3. J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel. The Web Service Modeling Language WSML. WSML Working Draft D16.1v0.21, available at: `http://www.wsmo.org/TR/`, Oct 2005.
4. M. Ehrig, S. Staab, and Y. Sure. Bootstrapping Ontology Alignment Methods with APFEL. *Proc of the 4th Intl Semantic Web Conf (ISWC-2005)*, Nov 2005.
5. J. Euzenat, F. Scharffe, and L. Serafini. Specification of the alignment format. Knowledge Web Deliverable D2.2.6, 2006.
6. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2), 1997.
7. M. Kerrigan, A. Mocan, M. Tanler, and D. Fensel. The Web Service Modeling Toolkit - An Integrated Development Environment for SWS. In *Proc of the 4th European Semantic Web Conf (ESWC-2007)*, Austria, Jun 2007.
8. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, (42):741–843, July 1995.
9. A. Mocan and E. Cimpian. An Ontology-Based Data Mediation Framework for Semantic Environments. *Intl Journal on Semantic Web and Information Systems (IJSWIS)*, 3(2), April - June 2007.
10. A. Mocan, M. Moran, E. Cimpian, and M. Zaremba. Filling the Gap - Extending Service Oriented Architectures with Semantics. In *Proc of the IEEE Int Conf on e-Business Engineering (ICEBE-2006)*, China, Oct 2006. IEEE Computer Society.
11. N. F. Noy and M. A. Munsen. The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping. *Intl Jrnl of Human-Computer Stud.*, 6(59):983–1024, 2003.
12. Y. Papakonstantinou, H. Garcia-Molina, and J. D. Ullman. MedMaker: A Mediation System Based on Declarative Specifications. In *Proc of the 12th Intl Conf on Data Engineering*, USA, 1996.
13. D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.
14. N. Silva and J. Rocha. Semantic Web Complex Ontology Mapping. In *Proc of the IEEE Web Intelligence (WI-2003)*, Canada, Oct 2003.