4. Balogh, A., Varro, D.: Advanced model transformation language constructs in the VIATRA2 framework. Symposium on Applied Computing Proceedings of the 2006 ACM symposium on Applied computing, Dijon, France, pp. 1280 – 1287, ISBN:1-59593-108-2 (2006)

5. Barzdins J., Barzdins G., Balodis R., Cerans K., Kalnins A., Opmanis M., Podnieks K.: Towards Semantic Latvia, Baltic DB&IS 2006, Communications, O.Vasileckas, J.Eder, A.Caplinskas (Eds). Vilnius: Technika, 2006. pp.203-218

6. UML® Resource Page, http://www.uml.org

7. OWL Web Ontology Language Guide, http://www.w3.org/TR/owl-guide/

8. OWL 2 Web Ontology Language Document Overview, http://www.w3.org/TR/owl2-overview/

9. Hart L., Emery P., Colomb B., Raymond K., Taroporewalla S., Chang D., Ye Y. and Dutra M., Kendall E.: OWL Full and UML 2 compared, March 2004

10. Marrying Ontology and Software Technology, European Commission ICT research 7th Research Framework Programme project, http://most-project.eu

11. Pereiras F.S., Staab S., Winter A.: On Marrying Ontological and Metamodeling Technical Spaces, ESEC/FSE'07, Dubrovnik, Croatia, pp. 439-448, ISBN:978-1-59593-812-1 (2007)

12. Brockmans S., Volz R., Eberhart A., Löffler P.: Visual Modeling of OWL DL Ontologies Using UML, S.A.McIlraith et al. (Eds.): ISWC 2004, LNCS 3298, pp.198-213, 2004

13. Brockmans S., Haase P., Hitzler P., Studer R.: A Metamodel and UML Profile for Rule-Extended OWL DL Ontologies, Y.Sure and J.Domingue (Eds.): ESWC 2006, LNCS 4011, pp.303-316,2006

14. Noy N.F., McGuiness D.L.: Ontology Development 101: A Guide to Creating Your First Ontology, http://protege.stanford.edu/publications/ontology_development/ontology101.html

15. ISO Released version of MOF (ISO/IEC 19502), v.1.4.1: Ch.7.9. MOF Model Constraints, http://www.omg.org/spec/MOF/ISO/19502/PDF/

16. Berners-Lee T.: Linked Data, http://www.w3.org/DesignIssues/LinkedData.html

17. OpenLink Virtuoso Universal Server: Documentation, http://docs.openlinksw.com/virtuoso/index.html

18. SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/

19. Schmidt M., Hornung T., Lausen G., Pinkel C.: SP2Bench: A SPARQL Performance Benchmark, arXiv: 0806.4627v2 cs.DB, Tech. Rep., 2008.

20. Schmidt M., Hornung T., Lausen G., Pinkel C.: SP2Bench: A SPARQL Performance Benchmark, http://www.informatik.uni-freiburg.de/~mschmidt/docs/sp2b.pdf

21. User Guide for Sesame 2.3 Chapter 9, http://www.openrdf.org/doc/sesame2/users/ch09.html

Semantic Technologies for Information Systems

RDB2OWL: Mapping Relational Databases into OWL Ontologies - a Practical Approach

Guntars Būmans[1,*], Kārlis Čerāns[2]

[1]Department of Computing, University of Latvia, Raiņa bulvāris 19, Riga LV-1586, Latvia
guntars.bumans@gmail.com
[2]Institute of Mathematics and Computer Science, University of Latvia
Raiņa bulvāris 29, Riga LV-1459, Latvia
karlis.cerans@mii.lu.lv

Abstract. In this paper we demonstrate a simple yet practically efficient approach for transforming source relational database (RDB) data into target OWL-structured RDF data format. We define a simple RDB2OWL mapping metamodel and provide its implementation by means of automatically generating SQL statements that create RDF triples corresponding to the target OWL ontology from source RDB data. We introduce and discuss possibilities to exploit meta-level information in mapping definition. The RDB2OWL mapping approach has been successfully applied in semantic re-engineering of Latvian medical registries data.

1 Introduction

With the advent of Semantic Web technologies and the need to enable those to access the massive amount of data that are existing in the form of relational databases (RDB) both in public domain and proprietary in companies and organizations, the need of bridging the RDB and semantic RDF/OWL data formats has become apparent and has been widely studied. The RDB to RDF data correspondence has been discussed already in [1]. Some of the most notable approaches dealing with RDB to RDF/OWL data mapping are R2O [2], D2RQ [3], Virtuoso RDF Views [4] and DartGrid [5]. There is W3C RDB2RDF Incubator Group [6] related to standardization of RDB to RDF mappings, the group has published its survey of mapping RDBs to RDF [7].

The mapping of RDB to RDF/OWL format has emerged as an issue of primary importance also in Semantic Latvia approach [8] and its application to the medical domain [9,10]. This approach proposes creating ontology (ontologies) for data that are available in a specific domain (e.g., government data, or medical data), using visual UML/OWL profile [11] or other related graphical notation, e.g. OWLGrEd [12,13], followed by data integration (mapping of existing, possibly legacy RDB data, into the format of the defined conceptual ontology), then followed by providing tools that are able to access the semantic data by means of a visual SPARQL query endpoint [9,14]. The initial RDB-to-RDF mapping definition by means of hand-coded SQL statements

[*]Partially supported by ESF project 2009/0138/1DP/1.1.2.1.2/09/IPIA/VIAA/004.

as outlined in [15] has appeared less than satisfactory in practice, as did the approach of hand-coding the mappings in a low-level model transformation language over the intermediate data representation forms in a MOF-based [16] repository.

The purpose of this paper is to offer a soundly motivated and practically efficient approach for RDB-to-RDF/OWL mapping that is suitable to cope with the motivating practical examples, as well as is extensible beyond those. Our solution, RDB2OWL, consists in a simple MOF-style mapping metamodel (that can be re-phrased easily also into a mapping OWL ontology), together with its implementation by means of automatically generating SQL statements that create (dump) RDF triples corresponding to the target OWL ontology from source RDB data. The simple structure of the metamodel allows treating its models also as documentation of the correspondence between the RDB and RDF/OWL schemas (accessible at least to technically literate user); this is important when the semantically re-engineered RDF/OWL models are themselves regarded as user-level documentation of the technical RDB schemas.

Although the basic concepts of the RDB2OWL mapping metamodel are nowadays fairly standard (a RDF/OWL class mapping to RDB table, a datatype property mapping to table field in the context of a mapping between the table and the property domain class, an object property mapping based on a relation between tables that are mapped onto the domain and range classes of the property, all of them with filter expressions and table joins, where necessary), their presentation here in a form of a metamodel appears to be less common. Furthermore, we identify a few typical mapping patterns and propose solutions for their transparent (user-friendly) encoding into a mapping definition, including the cases when this leads to "meta-level" operations over the RDB schema and/or OWL ontology definition (e.g., analyzing all properties with a fixed specified domain, necessary to succinctly reflect a conceptualization by means of subclasses; or meta-level information tables for grouping table fields into a single multi-valued datatype or object property). Yet another "non-common" point in RDB2OWL is "virtual" class-to-table mappings that do not generate class instances, but can be referred to in object or datatype property mappings.

We note that our approach is not (at least in this paper) looking for automated mapping generation from field-to-property correspondences in the style of CLIO [17] (on a practical note, we need a richer join filtering language than CLIO permits). We are not primarily looking at applying the defined mappings in retrieving the data from source RDB on-the-fly when the data are requested by queries in a RDF model environment, as in [3], [4] and [5]. This saves us at least the considerations for efficiency of integrating queries over RDB into those over RDF data stores, as well as allows for a greater freedom in mapping construction techniques. The closest approach to ours is that of R2O [2], where the same principal schema of employing the SQL engine for implementing the declaratively specified mappings is used.

The technical novelty of RDB2OWL with respect to these and related approaches is in the said mapping pattern observations which we believe to be worth considering in the RDB-to-RDF translation standardization effort [6]. On a practical side, we report on the experience of building RDB-to-OWL mapping for six Latvian medical registries [9], [10] within the presented simple mapping specification structure. We believe that the RDB2OWL mappings could be easily turned into a human readable form which is important when the target ontologies are regarded as documentation of the source RDB; this is an important factor within the Semantic Latvia [8] approach.

The rest of the paper is organized, as follows. Section 2 describes the overall setting for RDB2OWL mappings. Section 3 introduces the (core) RDB2OWL mapping metamodel and Section 4 illustrates it on a simple mini-university example taken from [15]. Section 5 describes the advanced mapping facilities, Section 6 reports on implementation and experience. Section 7 provides further related work notes and Section 8 concludes the paper.

2 The mapping framework

Figure 1 shows the architecture of RDB-to-RDF/OWL mapping process in the RDB2OWL framework.

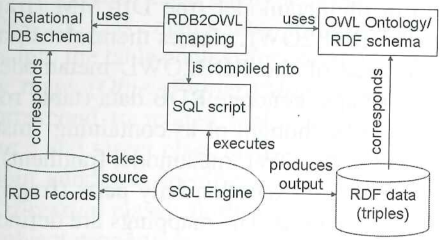

Fig. 1. RDB2OWL framework architecture

The task of the RDB2OWL mapping is to establish a correspondence between a relational database schema (or several schemas) and elements (entities) of OWL ontology (its schema part) or RDF schema (a single mapping can involve possibly several OWL ontologies), so that the corresponding RDB records could be translated (dumped) into RDF triples that correspond to the given ontology or RDF schema.

We assume that both the relational database and OWL ontology are given, with the typical situation in mind where the RDB contains an existing data set with technical metadata information and possibly de-normalization issues present, and OWL ontology contains a conceptual representation of the data present in the database (e.g., the classes and properties are provided with names understandable to a domain expert, a subclass relation between concepts is introduced where appropriate, a clear conceptual structure of the model is given precedence over technical structure optimizations possibly present in the RDB model).

A RDB2OWL mapping describes rules for extracting RDF triple information from RDB data on the basis of the source model properties – the properties that can be discovered within the RDB schema and the corresponding database records (e.g., table rows and fields therein, joining related tables, applying filters and transformations to data). We do not constrain the mapping generation process by applying OWL-based target model filters (e.g. cardinalities[1], property domain/range restrictions or presence of class axioms) over the generated triples. The OWL

[1] The cardinality imposition would entail the order-dependence of the triple generation process, since it may allow earlier instances to "get through", while blocking the "later" ones.

ontology constraints over the target RDF data can be checked independently after the completion of the mapping process and their satisfaction is regarded as an issue of the mapping correctness, and the semantic correctness of the source RDB data.

We note also that although we are primarily concerned with presentation of the RDB2OWL approach in conjunction with its implementation using an SQL engine, other mapping implementations on the basis of RDB2OWL are possible.

3 The mapping metamodel

We present the core RDB2OWL mapping metamodel in a MOF-style [14] in Figure 2, with additional expression and filter metamodel in Figure 3.

The metamodel refers to RDB schema and OWL ontology structure descriptions, presented here in the form of metamodel (the DB MM (fragment) and the OWL metamodel (fragment)). The RDB2OWL classes themselves are shown in the middle part of Figure 2. The instance of the RDB2OWL metamodel is a set of concrete mappings that each relates (maps) concrete RDB data (table row cell values) to RDF triples (these RDF triples can be thought of as containing "instance information" for the OWL concepts presented in the OWL metamodel fragment).

For the mapping definition and execution only parts (fragments) of RDB schema and OWL ontology structure are used. The mappings are defined in the terms of RDB tables and columns; they are not relying on primary key or foreign key information actually present in RDB schema (although, when accounted to, this information can be used to provide a more succinct mapping specification in a higher-level language). For the OWL part, in the case of raw mapping the only "structure" needed is URI associated to OWL entities (OWL classes, OWL datatype and object properties). For the advanced mapping features we include, however, the (optional) *subclassOf* information for OWL classes, as well as domain information for OWL properties and range information for OWL object properties.

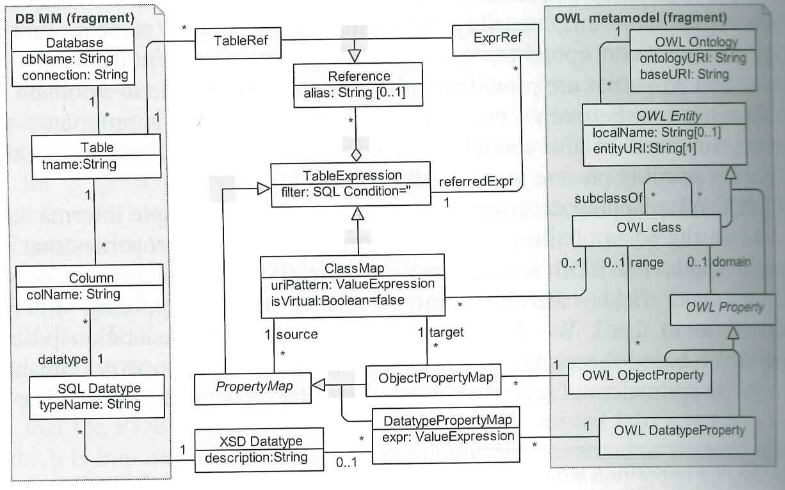

Fig. 2. The core RDB2OWL mapping metamodel

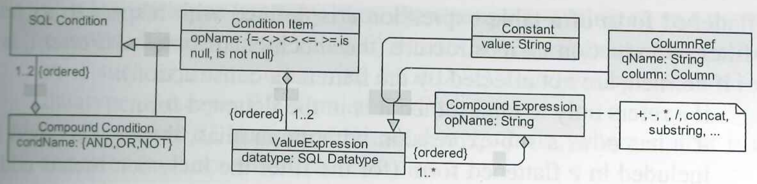

Fig. 3. The expression and filter metamodel

An RDB2OWL mapping consists of "elementary mappings", or maps, that are instances of *ClassMap*, *ObjectPropertyMap* and *DatatypePropertyMap* classes in the mapping metamodel. The class maps (*ClassMap* instances) are responsible for *Table-to-OWL Class* mappings (with options to add a filtering expression and linked tables).

The datatype property maps (*DatatypePropertyMap* instances) provide *Column-to-OWL DatatypeProperty* mappings. Each datatype property map is based on a source class map and can access the class map's table information; it can introduce further linked tables and filters into the table context for column expression evaluation.

The object property maps (*ObjectPropertyMap* instances) establish OWL object property links that correspond to related tables in the database. The tables to be related come from source and target class maps of the object property map; they are joined using explicit join condition specification in the object property map's *filter* attribute (from *TableExpression* class); further linked tables and filtering expressions can be introduced into object property maps, as well.

The class maps that are denoted as virtual (*isVirtual=true*) are not used for the RDF triple generation themselves, still, they can be referred to from object and datatype property maps. We require an instance of *OWL Class* to be linked to each non-virtual class map.

For an object property map x and its source and target class maps s and t we require that x as a table expression has an expression reference (an *ExprRef* instance) with alias 'S' that points to s as *referredExpr*, and an expression reference with alias 'T' that points to t. In a similar way, we require that a datatype property map (as a table expression) has a reference to its source class map (also, viewed as a table expression); in this case we allow not specifying alias name.

Informally, OWL classes *Teacher* and *Course* may relate to class maps that are based on DB tables *Teacher* and *Course*, respectively. An object property map for the property *teaches* can be described by a table expression *(Teacher S, Course T; S.Teacher_ID = T.Teacher_ID)*, where *S* is the alias for the source class map's table *Teacher* and *T* is the alias for the target class map's table *Course* and the string '*S.Teacher_ID = T.Teacher_ID*' is the value of the property map's *filter* attribute. Section 4 presents the example more precisely (see Figure 6).

Every table expression *e* (*TableExpression* instance; it can be also a class map or a property map) is built up from possibly alias-identified references that refer each either to a database table, or to another table expression (called a *sub-expression* of *e*); we require the sub-expression structure not to contain loops. Every table expression can be reduced to its *flattened form* that is a set of linked and possibly alias-identified tables that can be interpreted as a SQL table context for *filter* attribute, as well as for class map's *uriPattern* attribute and datatype property map's *expr* attribute evaluation.

The flattened form of a table expression *e* is defined with respect to its reference list and filter by induction on its structure (the subclass attributes *uriPattern*, *isVirtual* and *expr*, if defined, are not affected by the flattening construction):

(i) if *e* refers only to tables, then it is in the flattened form;

(ii) if *e* has *e'* as a sub-expression without an alias, then *e'* flattened form is included in *e* flattened form (for the filter the inclusion means adding the filter of e' flattened form as a conjunct);

(iii) if *e* has *e'* as a sub-expression with alias *A*, then *e'* flattened form *e'f* is included in *e* flattened form *ef* via adding the prefix *A* to all aliases that occur in *e'f*; if some table is included in *e'f* without an alias, it can be referred to by alias *A* in *ef*; the insertion of the alias or alias prefix *A* is performed also within the *e'f* filter before adding it to *ef* filter.

For instance, a join of two expressions *(Company C, Person P; C.CID=P.CID) S* and *(Person P, Address A; P.PID=A.PID) T* would be flattened into *(Company SC, Person SP, Person TP, Address TA; SC.CID=SP.CID and TP.PID=TA.PID)* that may be augmented at the outer level by a further filter, e.g. *SP.PID=TP.PID*.

We require that the flattened form of any table expression be unanimous as SQL table context and that the *filter* (its flattened form), *uriPattern* and *expr* attributes be defined within that table context.

3.1 Semantics of RDF triple generation

The RDF triple generation in accordance to a concrete RDB2OWL mapping and a concrete set S of RDB records that correspond to the given RDB schema is performed in two consecutive phases: (i) raw triple generation, and (ii) triple optimization.

The raw triple generation is performed for each class map and each object property map and each datatype property map separately, in any order.

For a non-virtual class map c, its flattened form is (as a table expression) evaluated against the RDB schema, filled by the records S, obtaining a set of (possibly joined and possibly repeating) data rows. For each of these data rows, the expression *c.uriPattern* is evaluated into a string value, denote it by *u*, and an RDF triple <*u*, 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type', *o.entityURI*> is formed.

In the case, if *c* flattened form has no references to DB tables, the *c.uriPattern* needs to be a constant value (or an expression thereof); in this case the class map *c* defines a constant RDF triple to be generated.

For a property map *p*, let *s* be the corresponding source class map. If *p* is an object property map, let *t* be the corresponding target class map. Let *o:OWL Property* be the one linked to *p*. Let *sa* (resp. *ta*) stand for the alias of the reference in *e* to *se* (resp. *te*).

We evaluate the property map *p* viewed as a table expression (we consider its flattened form, if necessary) against the RDB schema filled by the records S, obtaining a set of (possibly joined and possibly repeating) data rows. For each of these data rows the following actions are taken:

(i) the column references (*ColumnRef* instances) in the expression *s.uriPattern* are updated to include the *sa* prefix, and the updated expression *s.uriPattern*^{sa} is evaluated into a string value, denote it by *su*;

(ii) if *p* is a datatype property map, then *p.expr* is evaluated and the result is converted into a string value *d*; let *dt:XSD Datatype* be the one linked to *p*, if such *dt* exists, otherwise let *dt:XSD Datatype* be the one linked to datatype of *p.expr*. We form an RDF triple <*su, o.entityURI*, concat(*d*,'^^', *dt.description*)>, where concat is a string concatenation operation;

(iii) if *p* is an object property map, the column references (*ColumnRef* instances) in the expression *t.uriPattern* are updated to include the *ta* prefix, and the updated expression *t.uriPattern*^{ta} is evaluated into a string value, denote it by *tu*; we form an RDF triple <*su,o.entityURI,tu*>.

After the raw triple generation a simple triple optimization is applied (there are further optimizations described below in the extended mapping case), we call it "subclass optimization": for any two triples with the same subject <*u*, 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type', *URI_A*> and <*u*, 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type', *URI_B*>, if the OWL class with *entityURI* = *URI_A* is a subclass of OWL class with *entityURI* = *URI_B*, then the last of the said triples (the one relating *u* with the "superclass"), is dropped.

4 A mini-university example

Consider an example, adopted from [15] of a simple relational database schema reflecting a miniature study registration system and the related OWL ontology (Figure 4). The OWL ontology is presented in OWLGrEd [12,13] notation. Note that we do not focus on the integrity constraints in the OWL ontology here.

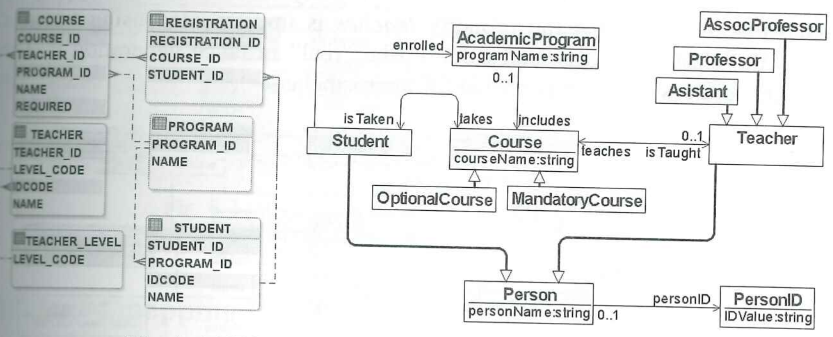

Fig. 4. A RDB schema and ontology of mini-university

Observe the table splitting (*Course*, *Teacher*) and table merging (*Person* from *Student* and *Teacher*) in the ontology using the subclass relations; OWL class *PersonID* that is based on non-primary key columns in each of *Student* and *Teacher* tables; and the n:n relation *takes* that reflects a student-to-course association that in the RDB is implemented using *Registration* table.

Figure 5 defines in abstract syntax via RDB2OWL metamodel instances all class maps for the mini-university example. Note that there are two class maps, generating instances of OWL class "PersonID".

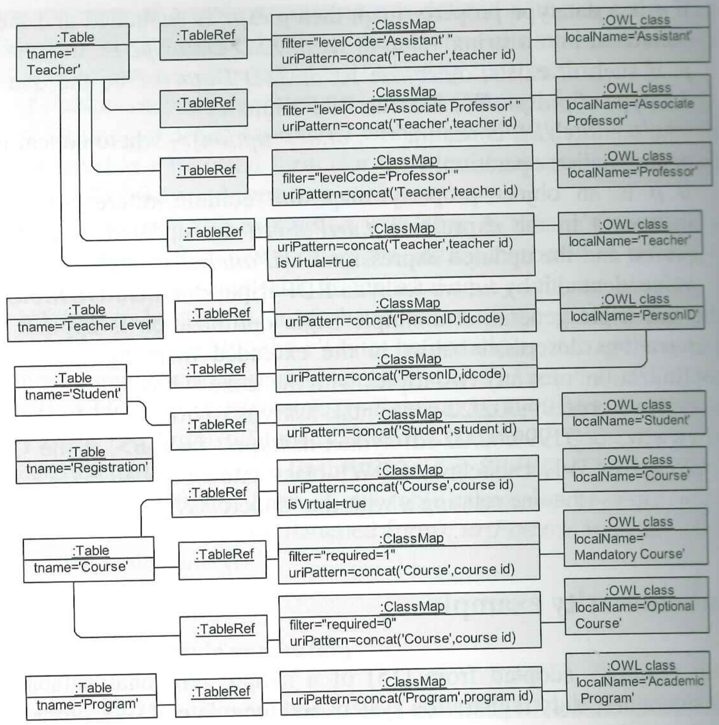
:Table
tname='Teacher'
:TableRef
:ClassMap
filter="levelCode='Assistant' "
uriPattern=concat('Teacher',teacher id)
:OWL class
localName='Assistant'
:ClassMap
filter="levelCode='Associate Professor' "
uriPattern=concat('Teacher',teacher id)
:OWL class
localName='Associate Professor'
:ClassMap
filter="levelCode='Professor' "
uriPattern=concat('Teacher',teacher id)
:OWL class
localName='Professor'
:ClassMap
uriPattern=concat('Teacher',teacher id)
isVirtual=true
:OWL class
localName='Teacher'
:Table
tname='Teacher Level'
:ClassMap
uriPattern=concat('PersonID,idcode)
:OWL class
localName='PersonID'
:Table
tname='Student'
:ClassMap
uriPattern=concat('PersonID,idcode)
:ClassMap
uriPattern=concat('Student',student id)
:OWL class
localName='Student'
:Table
tname='Registration'
:ClassMap
uriPattern=concat('Course',course id)
isVirtual=true
:OWL class
localName='Course'
:Table
tname='Course'
:ClassMap
filter="required=1"
uriPattern=concat('Course',course id)
:OWL class
localName='Mandatory Course'
:ClassMap
filter="required=0"
uriPattern=concat('Course',course id)
:OWL class
localName='Optional Course'
:Table
tname='Program'
:ClassMap
uriPattern=concat('Program',program id)
:OWL class
localName='Academic Program'

Fig. 5. Mapping instances: class maps

Figure 6 explains the instances for object property *teaches* and datatype property *courseName*. Observe that the property *teaches* is implemented using virtual class maps to the classes *Teacher* and *Course* (the "real" instance generation in Teacher and Course classes has been specified for their subclasses).[2]

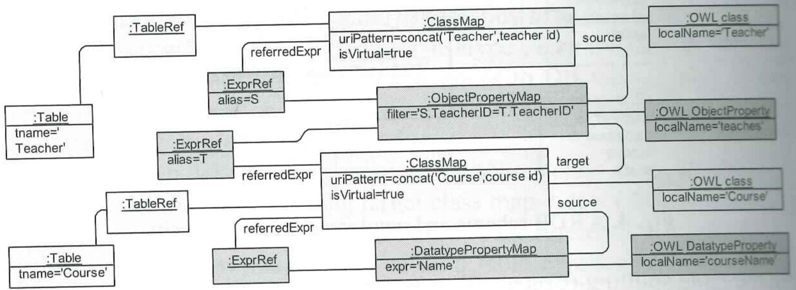

Fig. 6. Mapping instances: object property and datatype property maps

The "standard" solution to specification of the object property *takes* (going over the intermediate *Registration* table) is shown in Figure 7.

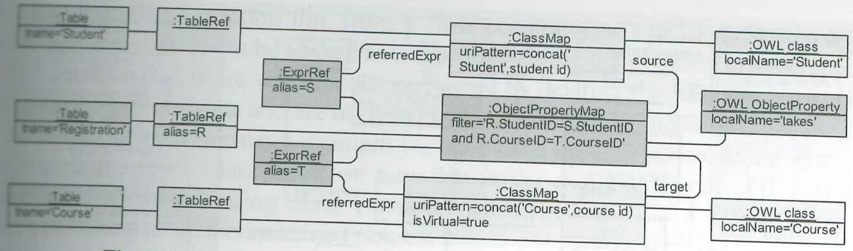

Fig. 7. Mapping instances: property mapping through linked table

There is, however, an alternative solution that is based on not using the "real" class maps for *Student* and *Course* table-to-class mappings, but defining virtual class maps for OWL class *Student* and *Course* URI generation directly from the *Registration* table (the *Registration* table contains the *Student_ID* and *Course_ID* columns that are needed for these URI generation). The alternative object property map definition is shown in Figure 8. The down side of this solution is the need to re-specify the URI patterns for subject and object URI generation, however this possibility outlines the power of the virtual class maps.

The virtual class maps in the style of Figure 8 are essential, if we want to define several RDB-to-RDF/OWL mappings, each of them responsible for a certain source database, and if we want to create some cross-database linking properties (e.g. on the basis of certain field value equality), where the mapping A can not access the instance-generating class map that is defined within the mapping B.

A full account of the mini-university example in RDB2OWL style, including the full instance generation explanation, is given in [18].

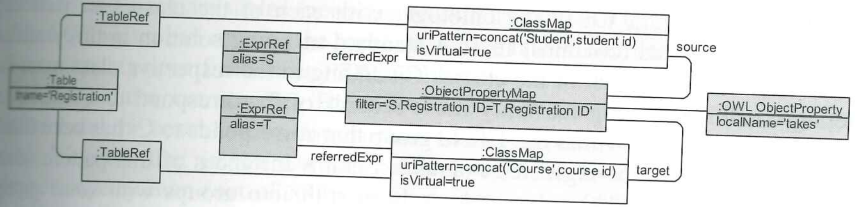

Fig. 8. Linking through table: virtual class maps

5 Advanced mapping facilities

There have been a few essential extensions to the core RDB2OWL mapping metamodel of Figure 2 and Figure 3 necessary for the successful mapping definition and implementation the Latvian medical registries case. These mapping patterns go beyond the classical class-to-table, data property-to-column, object property-to-table join pattern (with filtering and joining extra tables), as considered in Section 3. We believe that taking note of these patterns may be useful also for mapping definition in other cases, as well as in other mapping approaches. The extended RDB2OWL metamodel is given in Figure 9 (with the extension classes emphasized using bold frames). Yet further metamodel extensions are given in Figure 10 and Figure 12.

[2] The subclass optimization would achieve a similar effect also, if the class maps referred to in the *teaches* property map definition were not virtual. The virtual class maps allow achieving the needed triple set locally, without invoking the general subclass optimization principle.

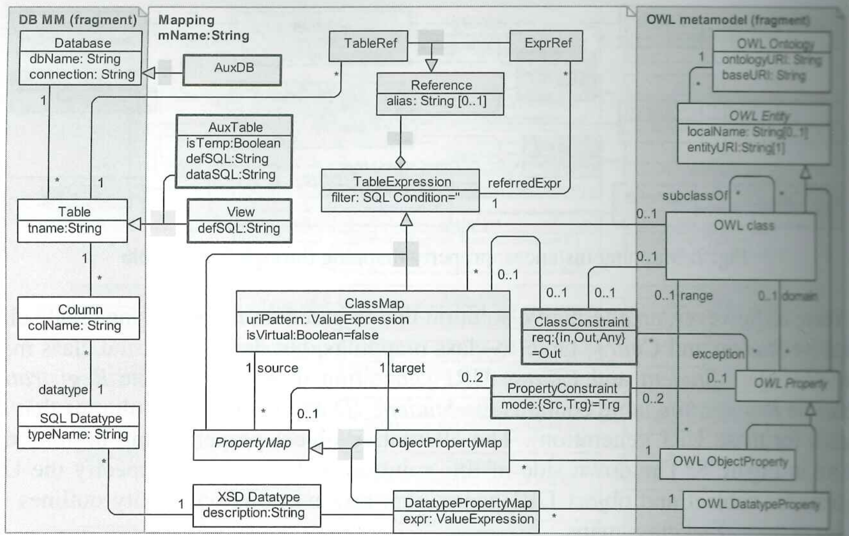

Fig. 9. Extended RDB2OWL mapping metamodel

5.1. Subclass conceptualization

One of the essential differences between the OWL ontology (conceptual model) and RDB schema is the use of subclass relation. By subclass conceptualization we understand a mapping pattern where a single database table, say T, corresponds to several classes C1, …, Cn in the ontology, with each of the classes Ci reflecting certain group of fields (columns) in T. A standard mapping solution in this situation would be to map T to each of the classes Ci, adding to the respective class maps the filtering conditions stating that only those records of T correspond to Ci instances where at least one field within the T field group that corresponds to Ci has been filled.

The Latvian medicine registries example contains instances of this pattern where tables with more than 200 columns have been split into groups with some groups responsible for more than 30 columns[3]. Given such a configuration the filtering conditions become lengthy, as well as difficult to write and especially to read.

We introduce into the mapping metamodel a *ClassConstraint* class whose instances denote the following requirement: generate <*x*,'rdf:type',*o*> triples only for those *x* that have a triple <*x*,*p*,*y*> for some property *p* with *p.domain*=*o* and some *y*.

A class constraint may be attached to an OWL class *o*, meaning that all its generated instances *x* (i.e., the <*x*,'rdf:type',*o*> triples) have to be checked for existence of property *p* instances for incoming (*p.range*=*o*), outgoing (*p.domain*=*o*, the default) or any (incoming or outgoing) properties. If a class constraint is attached to a class map, then it applies only to class instances that are created in accordance to this class map. The *exception* link from a class constraint, if specified, list datatype and object properties not to be looked at when determining the property existence.

The class constraints are semantically explained by deleting the RDF triples that do not satisfy its requirements and are obtained by the "raw triple generation".

In Latvian medical registries example we have used only class constraints that are specified on the OWL class level; we note that in the example 54 out of 172 OWL classes have a corresponding class constraint and 514 out of 814 OWL datatype properties belong to such constrained classes.

A *PropertyConstraint* class instance attached to a property *p* invokes the checking, if the subject *s* (*mode=Src*) or object *t* (*mode=Trg*) resources from a <*s,p,t*> triple have the triple <*s*,'rdf:type', *p.domain* > (or <*t*,'rdf:type', *p.range*>) generated by the mapping (or, present in the RDF triple store). Note that the property constraints with *mode=Trg* are not defined for datatype properties. The semantics of a *c:PropertyConstraint* is defined in terms of deletion of all property triples that do not satisfy *c* (after the *ClassConstraint* optimizations have been applied).

The use of such a property constraint may appear essential in conjunction with class constraint use (there is such a case in the sugar diabetes registry mapping), or it may be used to provide the constraints independently.

We stress that the class and property constraints are part of the mapping definition, not part of the target OWL ontology. The meaning of these constraints is fully "closed world": delete the triple, if the additional context is not created by the mapping.

5.2. RDB schema extension

The RDB2OWL mapping can include references to auxiliary databases, as well as SQL views and auxiliary tables (permanent auxiliary tables or temporary tables).

The SQL views can refer to the source database tables, and they can be used when a direct mapping specification as RDB2OWL core metamodel instance is either not possible or is not convenient. There are two such known cases: the aggregate functions[4] and the nested views construction (with computed columns in intermediate views). We note that in the Latvian medical registries example there is a single datatype property map (out of 832) whose definition would benefit from the nested view construction; there have been no need for grouping and aggregation.

The temporary tables can be used for holding intermediate computation results that are necessary in RDF triple generation in case of more complex RDB schema-to-ontology element correspondence. The full power of SQL can be used for temporary table population. We note that there have been no need to use temporary tables in Latvian medical registries example, however we foresee their use in other examples and therefore we offer a possibility of their usage smooth integration into mappings.

The permanent auxiliary tables are meant to be filled during the mapping specification (the data in these tables can be conceptually thought to be a part of the mapping definition). There are permanent auxiliary tables of two kinds used in the medicine registries example: a *Numbers* table, containing the only column *N* storing the natural numbers from 1 up to 1000, and a *NewCodes* table (in fact, several tables

[3] There are different groups of measurements taken during a clinical anamnesis, all measurements are recorded into a single table; however, the conceptual model indicates the different groups to which the measurements belong to.

[4] It would not be difficult to add aggregate functions directly into RDB2OWL metamodel.

of this kind) to store classifiers which have been identified during the ontology definition process and which have not been recognized as classifiers in the relational database (in Section 5.4 the *DTreatment* table is of *NewCodes* table kind).

Joining a data table to linked *Numbers* table is useful in situation when a table field value is to be split into multiple datatype property values. Let *A* be a table with a column *ManyYears* whose values are non-separated 4-digit year sequences (e.g. '198820022005'). The datatype property map *(A, Numbers; len(ManyYears)<N*4; expr=substring(ManyYears,(N-1)*4,4))* splits each of these values into the appropriate number of datatype property assertions, each referring to a single year within the year sequence. We note that this mapping pattern is used for different table columns in 111 datatype property maps in Latvian medicine registries example.

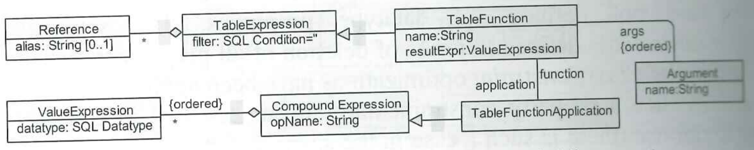

Fig. 10. Table functions (a mapping metamodel fragment)

5.3. Table functions

The table functions (see Figure 10) allow wrapping adding a fixed linked table, a filtering expression and datatype property value expression into a single function call. In case of value splitting with *Numbers* table from previous section, a function *split4* (an instance of *TableFunction* class) could be defined to refer (as a table expression) to *Numbers* table, it could have a single argument X, the *filter=len(X)<N*4* and *resultExpr=substring(X,(N-1)*4,4)*. The usage of the *split4* function in the above example then would amount to the simple call *split4(ManyYears)*.

The semantics of the use of *split4(Q)* in a value expression *v* in the context of some table expression *e* involves the extension of *e* context by the *Numbers* table (an expression *e1* that has *e* as its sub-expression has to be created, the name prefixes have to be adjusted to avoid clashes), and evaluating of *v* with *Q* substituted for the argument *X* in this extended context. The precise function application constraints, although intuitively clear, are yet to be precisely spelled out.

The table functions can be defined also for e.g. "classifier value" lookup in the source database, thus possibly allowing creating substantially shorter mapping specifications in the databases that rely on the classifier encoding.

5.4. Using meta-level data

Figure 11 informally depicts a mapping between a database table *PatientData* and an OWL class *PrescribedTreatment*, where five Boolean fields of the database table are mapped each onto an instance of property *diabetesTreatment*. An instance of the *PrescribedTreatment* OWL class has a *diabetesTreatment*-link to the instance of *DiabetesTreatment* with *description*=TreatmentX (X stands for A,B,C,D or E) whenever the *TreatmentFieldX* value in the corresponding *PatientData* table record is 1.

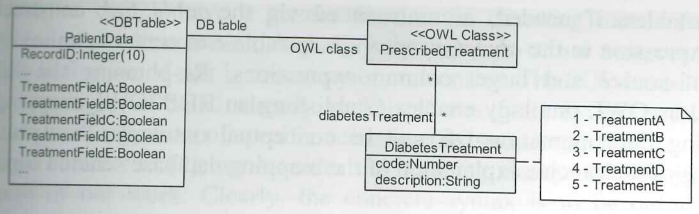

Fig. 11. A "field grouping" mapping pattern example

We create an auxiliary table *DTreatment(Code,Value)* and place the data {(1,TreatmentA), (2,TreatmentB), (3,TreatmentC), (4,TreatmentD), (5,TreatmentE)} therein, as well as create a class map that maps the *DTreatment* table into the *DiabetesTreatment* OWL class. In the "standard" approach, five object property maps are created one for each of *TreatmentFieldX* columns, creating the *diabetesTreatment*-links into *DiabetesTreatment* instance with *description*=TreatmentX from corresponding *PrescribedTreatment* instances. An example filter here is '*TreatmentFieldD=1 and code=4*' (the conditions are similar for other four cases, as well).

An approach based on using meta-level data involves creating an auxiliary table *MDTreatment(Code,Value)*, where the *PatientData* table column names are recorded, and linked with the codes of respective *DiabetesTreatment* instances. The table data become: {(1,TreatmentFieldA), (2,TreatmentFieldB), (3,TreatmentFieldC), (4,TreatmentFieldD), (5,TreatmentFieldE)}. The single object property map for *diabetesTreatment* is then built as follows: *(PatientData S, DTreatment T, MDTreatment M; lookup(S,M.Value)=1 and T.Code=M.Code)*.

The semantics of the user defined function *lookup(A,C)* with *A* being a table name and *C* a column, in the context of a particular table A record ***R*** is to return the value of that field in ***R*** whose name is stored in *C*. Such a function can be easily implemented e.g. using SQL engine means (the evaluation of a user-formed SQL string).

In the RDB2OWL metamodel the *lookup*-expressions are defined to form a new subclass of *CompoundExpression* class (Figure 12).

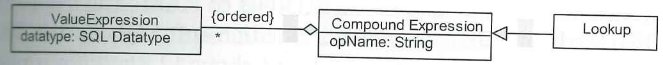

Fig. 12. Meta expressions (lookup).

6 Implementation and experience

The implementation of RDB2OWL mapping language has been based on mapping data themselves stored in a RDB schema (we call it mapping DB schema). Figure 13 shows the actual mapping DB schema implemented in our approach. The current implementation supports only single table class maps as well as class constraint and property constraint definition only on OWL class and OWL object property level. The support for table functions and meta-level data usage is also under development.

In the implementation schema the datatype and object property maps automatically "see" the tables referred in their source and target class maps; the additional tables, if needed, are introduced via the *table_link* construction. The filtering expression in the *object_property_map* table can contain explicitly specified equality of source and target column expressions. Re-phrasing the RDB2OWL metamodel as OWL ontology enables establishing an RDB2OWL mapping between the mapping implementation DB and its conceptual ontology, thus providing the methodology for a precise explanation of the mapping database schema constructs.

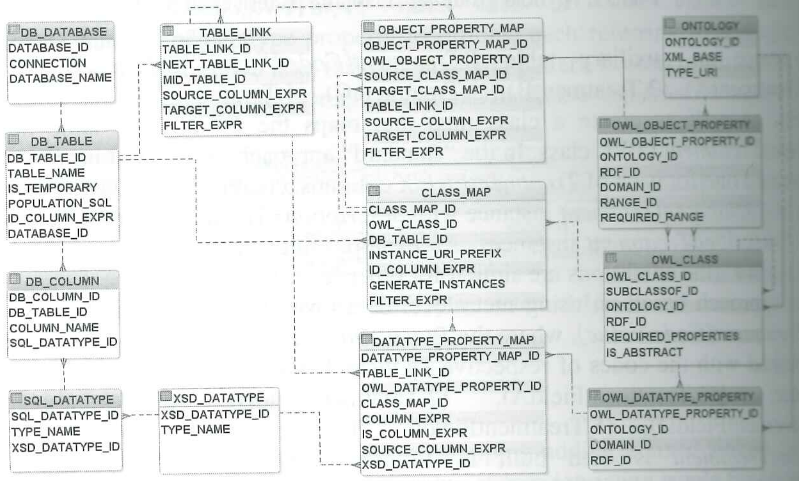

Fig. 13. The mapping DB schema

The data mapped from RDB to OWL format consists of 6 Latvian medical registries [9],[10], including 106 source database tables, 1353 table columns and total more than 3 million rows, altogether 3 GB of data. The corresponding OWL ontology had 172 OWL classes, 814 OWL datatype properties and 218 OWL object properties.

The mapping has been implemented on a laptop computer with Intel Mobile Core 2 Duo T6570 processor running Windows Vista, 3 GB of RAM. The mapping DB as well as source DB (Medicine DB) was served by Microsoft SQL Server 2005. The triple generation process from source DBs produced about 42.8 million triples and it has taken 27.5 minutes, out of which 9.5 minutes for raw triple generation, 8 minutes for indexing, 4 minutes for *ClassConstraint* enforcement and 6 minutes for triple exporting from table to text files in N-TRIPLE format (total file size 3.4GB).

7 Related Work

The core constructs of RDB2OWL mapping language are in one or other form common to most of the nowadays used RDB-to-RDF/OWL mapping approaches, including R2O [2], D2RQ [3], Virtuoso RDF Views [4] and DartGrid [5]. We acknowledge that most of these approaches (e.g., [3], [4], [5]) aim at a "higher" goal of retrieving the data from source RDB on-the-fly when the data are requested by queries in a RDF model environment. This has not been a requirement in our case, therefore we, as in [2], have a greater freedom in choosing means for mapping specification and implementation; including the freedom to use temporary tables in the mapping definition, if that is necessary or convenient. The SQL-based implementation of mappings, as noted already in [2], allows for easy extensibility of the mapping solution in the case of specific needs in constructs in the future.

In a slight contrast to [2], we consider the mapping clarity and simplicity to be a primary aim of our work. Clearly, the concrete syntax is to be defined for that, however the concise metamodel-based mapping structure presentation, as well as the concrete advanced mapping specification constructs we are proposing shows our striving for concise, well structured and readable mapping specifications.

It should be possible to translate at least a large portion of the RDB2OWL mapping specification constructs e.g. into D2RQ or Virtuoso RDF Views notations (there may be difficulties translating temporary table usage), however, we expect a significant mapping size increase at least in the case of subclass conceptualization constructs (the appropriate filter expressions are to be automatically generated during the translation).

8 Conclusions

In this paper we have demonstrated a simple and practice-oriented approach to mapping relational database data into RDF/OWL format, by offering a MOF-style metamodel for declarative specification of mappings and implementing the mappings by translating them into SQL scripts that are executable by an SQL Engine. We reported also on the successful application of our mapping approach in semantic re-engineering of Latvian medical registries.

We have outlined a few mapping construction patterns that would be good to have supported in RDB-to-RDF/OWL data mapping/integration frameworks and that have attracted little attention elsewhere. The subclass conceptualization can be mentioned here as well as mappings using source DB schema elements on the object level.

The further work is to develop user-friendly concrete syntax of mapping specifications on the basis of the RDB2OWL framework, with the aim of making the RDB-to-OWL mapping specifications easily accessible at least to a technically minded person and to the extent possible also to subject matter experts. Given the clear structure of the created RDB2OWL metamodel we deem this to be a feasible task.

Our primary interest is to develop RDB2OWL as a readable mapping specification language, still it would be interesting to see, how our approach could benefit from automated mapping discovery techniques, presented e.g. in [17] and [19].

There may be approaches other than SQL script generation for the RDB2OWL mapping implementation. We have noted a possibility of translating RDB2OWL mappings into other RDB-to-RDF/OWL bridging approaches. An interesting work would be also translating RDB2OWL mappings into MOF-style model transformations using a higher-order model transformation language, e.g. Template MOLA [20].

Acknowledgements. We would like to thank Sergejs Rikačovs for providing us with a detailed explanation of the RDB-to-RDF/OWL mapping requirements for the Latvian medicine registries.

References

1. T.Berners-Lee: Relational Databases on the Semantic Web, http://www.w3.org/DesignIssues/RDB-RDF.html, 1998.
2. J.Barrasa, O.Corcho, G.Shen, A.Gomez-Perez: R2O: An extensible and semantically based database-to-ontology mapping language. In: SWDB'04, 2nd Workshop on Semantic Web and Databases, 2004.
3. D2RQ Platform. http://www4.wiwiss.fu-berlin.de/bizer/D2RQ/spec/
4. C.Blakeley: "RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping)", OpenLink Software, 2007.
5. W.Hu, Y.Qu: Discovering Simple Mappings Between Relational Database Schemas and Ontologies", In Proceedings of 6th International Semantic Web Conference (ISWC 2007), 2nd Asian Semantic Web Conference (ASWC 2007), LNCS 4825, pages 225-238, Busan, Korea, 11-15 November 2007.
6. W3C RDB2RDF Incubator Group, http://www.w3.org/2005/Incubator/rdb2rdf/
7. A Survey of Current Approaches for Mapping of Relational Databases to RDF, http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf
8. J.Barzdins, G.Barzdins, R.Balodis, K.Cerans, et.al.: (2006). Towards Semantic Latvia. In Communications of 7th International Baltic Conference on Databases and Information Systems, pp.203-218.
9. G.Barzdins, E.Liepins, M.Veilande, M.Zviedris: Semantic Latvia Approach in the Medical Domain. Proc. 8th International Baltic Conference on Databases and Information Systems. H.M.Haav, A.Kalja (eds.) Tallinn University of Technology Press, pp. 89-102. (2008).
10. G.Barzdins, S.Rikacovs, M.Veilande, and M.Zviedris: Ontological Re-engineering of Medical Databases, Proceedings of the Latvian Academy of Sciences. Section B, Vol. 63 (2009), No. 4/5 (663/664), pp. 20–30.
11. Ontology Definition Metamodel. OMG Adopted Specification. Document Number: ptc/2007-09-09, November 2007. http://www.omg.org/docs/ptc/07-09-09.pdf
12. J.Barzdins, G.Barzdins, K.Cerans, R.Liepins, A.Sprogis: OWLGrEd: a UML Style Graphical Editor for OWL, to appear in Proceedings of ORES 2010, ESWC 2010 Workshop on Ontology Repositories and Editors for the Semantic Web, 2010.
13. OWLGrEd, http://owlgred.lumii.lv/
14. G.Barzdins, S.Rikacovs, M.Zviedris: Graphical Query Language as SPARQL Frontend. In Grundspenkis, J., Kirikova, M., Manolopoulos, Y., Morzy, T., Novickis, L., Vossen, G. (Eds.), Local Proceedings of 13th East-European Conference (ADBIS 2009), pp. 93–107. Riga Technical University, Riga, 2009.
15. G.Barzdins, J.Barzdins, K.Cerans: From Databases to Ontologies, Semantic Web Engineering in the Knowledge Society; J.Cardoso, M.Lytras (Eds.), IGI Global, 2008 (ISBN: 978-1-60566-112-4) pp. 242-266
16. OMG's MetaObject Facility, http://www.omg.org/mof/
17. R.Fagin, L. Haas, M. Hernandez, R. Miller, L. Popa, Y. Velegrakis: Clio: Schema Mapping Creation and Data Exchange. In Conceptual Modeling: Foundations and Applications, 2009.
18. G.Bumans, Mapping between Relational Databases and OWL Ontologies: an Example, to appear in Scientific Papers of University of Latvia, Computer Science and Information Technologies, 2010.
19. Y.An, A.Borgida, J.Mylopoulos: Inferring complex semantic mappings between relational tables and ontologies from simple correspondences. In: OTM'05, On The Move Federated Conference, 2005.
20. E.Kalnina, A.Kalnins, E.Celms, A.Sostaks: Graphical template language for transformation synthesis. In: M. van den Brand, D.Gašević, J.Gray (Eds.), Proceedings of Second International Conference, SLE 2009, Denver, CO, USA, October 5-6, 2009 Revised Selected Papers, LNCS 5969, pp. 244--253. Springer, Heidelberg, 2010.

Spatial Ontology in Statistical Machine Translation

Raivis Skadiņš,
Tilde, Vienibas gatve 75a,
Riga, Latvia
Raivis.Skadins@tilde.lv

Abstract. This paper presents a statistical phrase-based machine translation system which is enriched with semantic data coming from a spatial ontology. Paper presents the spatial ontology and how it is integrated in statistical machine translation system using factored models. Spatial information is added as a factor in both translation and language models. SOLIM spatial ontology language is used to implement ontology and to infer necessary knowledge for training statistical machine translation system. The machine translation system is based on Moses toolkit.

Keywords: Spatial Ontology, Statistical Machine Translation, RCC8.

1 Introduction

There are several different state-of-the-art approaches for building Machine Translation (MT) systems. Most popular methods used are knowledge based methods and corpus based methods. First MT systems where knowledge or rule based MT systems. They use human created rules, dictionaries and knowledge bases to describe grammars of languages and to specify translation rules. Recently corpus based (statistical and example based) MT methods are getting more and more popular. Corpus based MT systems do not need human created rules, grammars and dictionaries; they use statistical methods to analyze large amounts of human translated texts. Statistical MT (SMT) systems extract statistical data about translations and use these data to translate new texts.

Both knowledge and corpus based methods have their advantages and disadvantages. Knowledge based MT systems perform well if they have all necessary knowledge. Natural languages are very complex with many exceptions, uncertainties and ambiguities in rules and dictionaries. Therefore it is not possible to provide all necessary knowledge for knowledge based MT systems and these systems typically are working with incomplete data and that leads to many mistakes. It is possible to improve knowledge based MT systems only up to the certain quality level and further improvements get too complex and take too much human work.

Corpus based MT systems automatically extract statistics from huge amounts of translated texts (parallel corpus); they do not need human created rules and dictionaries. Just more and more training data are needed to improve corpus based MT systems. The main weakness of corpus based MT methods is the availability of reasonably sized parallel corpus. There are parallel corpora available for world's major languages such as English, French, Spanish, Chinese and Arabic, but smaller languages are often lacking enough parallel corpora to build good quality SMT systems. Pure SMT methods [1] [2] are not using any linguistic information including information about morphology and word inflections. As a result of that pure SMT systems perform better working with lesser inflected languages, such as English, but they perform much worse working with highly inflected languages, such as Baltic and Finno-Ugric languages, and more training data are needed to achieve the same level of MT quality for highly inflected languages as for lesser inflected languages.

The modern SMT methods are using different additional knowledge to build more sophisticated statistical models thus improving MT quality. Typically morphological and syntactical annotations are added to training data, and SMT systems learn translation and language models which benefit from this annotation. Typical examples of such SMT methods are factored phrase-based SMT [3], tree-based SMT [4] [5] [6], treelet SMT [7].

Rule based MT systems use rules and knowledge in different levels of analysis, some systems deal only with morphology and shallow syntax, others use also deep syntactic analysis and some systems use even semantic analysis. SMT systems currently are using only morphologic and syntactic information.

This paper presents ongoing research which is aiming to add some semantic knowledge to SMT. MT systems could benefit from various kinds of semantic knowledge in various stages of translation or training processes. Semantic information might be used in word breaking, part-of-speech tagging, syntactic disambiguation, word sense disambiguation etc. All mentioned areas are clearly distinguished in rule based MT systems, but they are somewhat vague in SMT. There is no words sense disambiguation component in SMT, but SMT models are built so that they can deal with word ambiguities. Although various kinds of semantic knowledge could be used to improve various translation aspects, such as word sense disambiguation, translation selection or phrase reordering, this paper is focusing on using of spatial information for word sense disambiguation in factored phrase-based SMT. Spatial ontology language developed in SOLIM[1] project is used to implement spatial.

In addition to this introductory chapter, this paper is divided into three main chapters. In chapter two, a brief introduction to SOLIM ontology language and implementation of the ontology is presented. Chapter three presents integration of the spatial ontology in SMT system. In chapter four current results are presented and further research is outlined.

2 SOLIM Language and Ontology Implementation

Spatial information, as well as information in general, is mostly implicit in nature. Its character is descriptive and meaningless without its most skilled interpreter: the human brain. If we want to use spatial information to train better SMT models, then this information should be described in a machine-readable way that enables artificial agents to at least act as if they would have some understanding of the information

[1] SOLIM project. EUREKA's Eurostars Programme. no.: E! 4365. http://www.solim.eu

being processed. The Web Ontology Language (OWL), a W3C standard, is a state-of-the-art means to achieve this.

OWL provides the means to go beyond keywords and specify the meaning of the resources described. It does so by imposing a well-defined structure on OWL documents and by providing the needed vocabulary and semantics for achieving the above-named goal. In short, OWL can be described as a means of specifying at least some of the entities in a world and the relationships between them. The processing of the data and any other process involving some intelligent aspects is done by applications external to the ontology, such as reasoner.

However, not all concepts are equally easy to define. One of the concepts that are limited by OWL's structure is space. Spatial relations and spatial reasoning are required for many purposes, such as geographical information systems, spatial databases, etc. And spatial information is important also for word sense disambiguation in MT. The problem is not so much that OWL is not capable of handling spatial representations, but rather that it is not designed for this type of encoding or reasoning and this makes it rather difficult to express some spatial properties and relationships and to be able to reason with these concepts.

Since the worlds described in OWL are a reflection of the real-world around us, concrete concepts in OWL implicitly have a spatial property. Yet, the OWL language does not allow expressing the spatiality of those properties in a useful way. This is the goal of the SOLIM project: to expand the OWL langue to allow for meaningful explicit expression of spatial properties.

This paper describes the set of essential features that the SOLIM language presents. There are two types of aspects regarding spatiality - the absolute location in space, in the form of coordinates, geo references etc. and the relative spatiality, i.e. entities of which the location is only defined as binary properties between concepts or their instances. As geographical coordinates are not typical for texts which are translated using MT systems, relative aspect of spatiality is more important for MT systems.

When reasoning with and about spatial properties, the question is raised what the notions of space or spatiality mean. Even though an exhaustive answer to this question is far beyond the purpose of this paper, a practical definition of spatiality is needed.

A first aspect of space relevant for the SOLIM language is that it relates to the representation of both absolute and relative locations and a second aspect is that it can represent dimensions (size, shape, etc). Such basic definitions of space in OWL would enable it to represent the constraints and properties of objects and is not explicitly supported in the version 1.0 of OWL-DL. The SOLIM project defines a language that supports the definition of both absolute and relative spatial properties, and – as a result – enables the representation of Region Connection Calculus (RCC) relations.

There are many types of spatial relationships, and an often used set for representing spatial relationships is the RCC-8 calculus. However, RCC-8 is not easily implemented in ontology languages because it requires certain adaptations of them. The goal of SOLIM project is to extend the language of choice; OWL, so that it can represent RCC-8 relationships properly. This involves an adaptation of the OWL language and the reasoner.

In considering possibilities and requirements for a comprehensive ontological scheme for spatial representation, it is essential to incorporate the very extensive work that has been carried out primarily within the tradition of qualitative spatial representation. As a starting point for discussion we take two independently developed views of spatial relationships: the Region Connection Calculus, RCC, proposed by Randell, Cui & Cohn [8] and the set of topological constraints proposed by Egenhofer [9]. These involve stating basic spatial relationships that may hold between spatial entities and working out ways of both reasoning with them and applying them to complex spatial configurations. Although Egenhofer describes his relations in purely topological terms and draws on set theory (regions as sets of points) for definitions, while Randell et al. draw on a topology of regions with spatial parts and start from the connection relation alone, there are clear similarities between them. Their ontological commitments are, however, somewhat different concerning the particular kinds of spatial objects assumed. The relations proposed are set out in Fig. 1. with the names employed in both approaches.

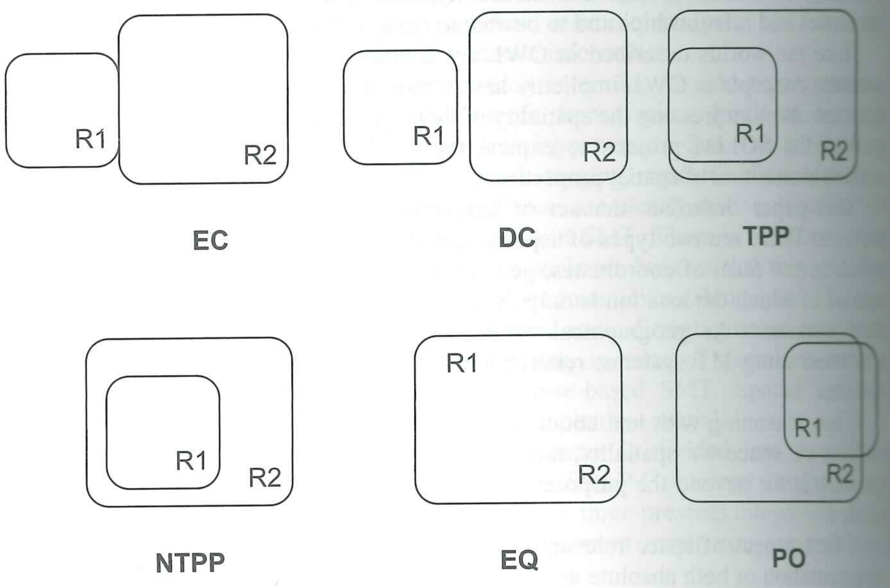

Fig. 1. The standard 'base relations' of RCC and similar calculi: disjoint/disconnected (DC), meet/external connection (EC), overlap/partial overlap (PO), equal (EQ), covered by/tangential proper part (TPP), inside/non-tangential proper part (NTPP). The inverses of the latter two are not shown: covers/tangential proper part inverse (TPP−1), contains/non-tangential proper part inverse (NTPP−1).

When formalizing them, both approaches make central use of the topological relationship of connection and typically begin their accounts with an enumeration of the distinct ways in which spatial entities can be related spatially: thus leading to these standard eight relations. There are, however, a number of ways of formalizing this view of spatial entities and their relations. While the mereotopological approach common in ontology begins by axiomatizing the relations and their properties in a first-order logical language (in terms of connection and parthood), the computational behavior of such formalization is not good because of its expressiveness.

The full first-order theory of RCC inherits undecidability and so various, more constrained, adaptations of the full theory have been defined. Within qualitative spatial reasoning, therefore, formalizations of spatial relationships that draw on the formal properties of other mathematical accounts is a very active area. By this means, one attempts to achieve more attractive computational properties that are more amenable to practical reasoning tasks. This also appears motivated cognitively in that certain tasks are readily performable by humans and others less so; such evidence can further constrain the kinds of modeling decisions that may eventually be made in formulating a realistic ontology of space.

As was described by Katz [10], RCC8 logic can be expressed in OWL DL with the exception of reflexive roles. Reflexive roles are required since the main property required to express RCC relationships – the "connects" property – has to be reflexive as any region is connected to itself. With the emergence of OWL 2.0, reflexive roles are available and therefore OWL2 is sufficiently expressive to allow the formal description of RCC relations.

To express RCC8 relationships the following translations (see formulas 1 to 6) are made:

$$EC(X,Y) \rightarrow \begin{cases} \forall R.X \sqsubseteq \exists R.\neg Y \\ Z_5 \equiv X \cap Y \end{cases} \tag{1}$$

The EC relationship is slightly more complex than for example DC (see formula 2) or EQ (see formula 5) as it involves the creation of two named classes (R and Z_5). These classes are defined as follows:

1. the class consisting of the points that only connect to X (the interior of X)
2. the class consisting of the points that have at least one connection to the region outside Y (the complement of the interior of Y).

Furthermore, the first of these two classes is expressed to subsume the second. Thus, to express external connectedness, the region outside Y has to contain the interior of X.

$$DC(X,Y) \rightarrow X \sqsubseteq \neg Y \tag{2}$$

$$TTP(X,Y) \rightarrow \begin{cases} X \sqsubseteq Y \\ Z_1 \equiv X \cap \exists R.\neg Y \end{cases} \tag{3}$$

$$NTTP(X,Y) \rightarrow X \sqsubseteq \forall R.Y \tag{4}$$

$$EQ(X,Y) \rightarrow X \equiv Y \tag{5}$$

$$PO(X,Y) \rightarrow \begin{cases} Z_2 \equiv \forall R.X \cap \forall R.Y \\ Z_3 \equiv X \cap \neg Y \\ Z_4 \equiv \neg X \cap Y \end{cases} \quad (6)$$

Ontology implemented in SOLIM language allows representing spatial information about different object and inferring implicit spatial information. Currently SOLIM project partners are evaluating suitability of SOLIM language for machine translation and for image search. The current implementation of ontology for machine translation contains spatial information about all continents, countries and major cities in English, Latvian and Lithuanian. Reasoner can use the ontology to infer spatial relations between objects. Reasoner has web service interface allowing external applications to query it using SPARQL query language.

3 Integration of Spatial Ontology into SMT

The proposed MT system is based on statistical machine translation, where translations are generated on the basis of statistical models whose parameters are derived from the analysis of bilingual and monolingual text corpora. The process of statistical model generation is called training, and typically involves analyzing vast amounts of parallel sentence pairs in two languages (bilingual corpus) in order to generate a translation model, as well as analyzing the target language (monolingual corpus) in order to generate a language model.

The translation model represents a certain statistical model of how a source language (L1) is translated into a target one (L2). The translation model can operate in the domain of words, phrases and more complex structures (for example, syntax trees), and in general is responsible for the adequacy of translation. The language model represents the knowledge about the target language, its sentence and phrase structure and is in general responsible for the fluency of translation.

The current state-of-the-art approach to SMT, so-called phrase-based models, are limited to the mapping of small text chunks (phrases) without any explicit use of linguistic information, may it be morphological, syntactic, or semantic. Such additional information has been demonstrated to be valuable by integrating it in pre-processing or post-processing. However, a tighter integration of linguistic information into the translation model is desirable for two reasons:

1. Translation models that operate on more general representations, such as lemmas[2] instead of surface forms of lexical units, can draw on richer statistics and overcome the data sparseness problems caused by limited training data.

2. Many aspects of translation can be best explained on a morphological, syntactic, or semantic level. Having such information available to the translation model allows the direct modeling of these aspects. For instance: reordering at the sentence level is mostly driven by general syntactic principles, local agreement constraints show up in morphology, etc.

[2] Lemma is the canonical form, dictionary form, or citation form of a particular word (surface form), for example 'read' is a lemma of surface form 'reading'.

Moses [11] is a statistical machine translation system that allows automatic training of translation models for any language pair. All you need is a collection of translated texts (parallel corpus). Moses has the following distinct features:

- Beam-search: an efficient search algorithm finds quickly the highest probability translation among the exponential number of choices;
- Phrase-based: the state-of-the-art in SMT allows the translation of short text chunks;
- factored: lexical units may have factored representation (surface forms, lemma, part-of-speech, morphology, lexeme classes...).

Moses framework is an extension of the phrase-based approach. It adds additional annotation at the lexical unit level. A lexical unit in our framework is not anymore only a token, but a vector of factors that represent different levels of annotation. The training data (a parallel corpus) has to be annotated with additional factors. For instance, if it is necessary to add part-of-speech information on the input and output side then part-of-speech tagged training data are required. Typically this involves running automatic tools on the corpus, since manually annotated corpora are rare and expensive to produce.

The proposed SMT system is based on the Moses SMT system, which features factored translation models that allow integrating additional layers of data tightly into the process of translation. The translation process translates the source (input) text into the target (output/translation) using the trained language and phrase (translation) models (see Fig. 2.). The system output is then compared with a reference (human) translation to evaluate the translation quality using automated metrics, such as BLEU score [12].

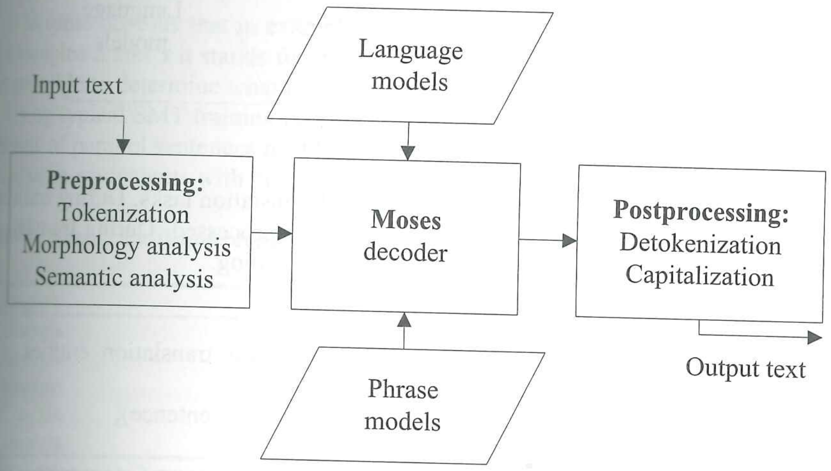

Fig. 2. Translation process.

The training process (see Fig. 3.) involves processing a parallel training corpus to generate the phrase model and a monolingual training corpus to generate the language model.

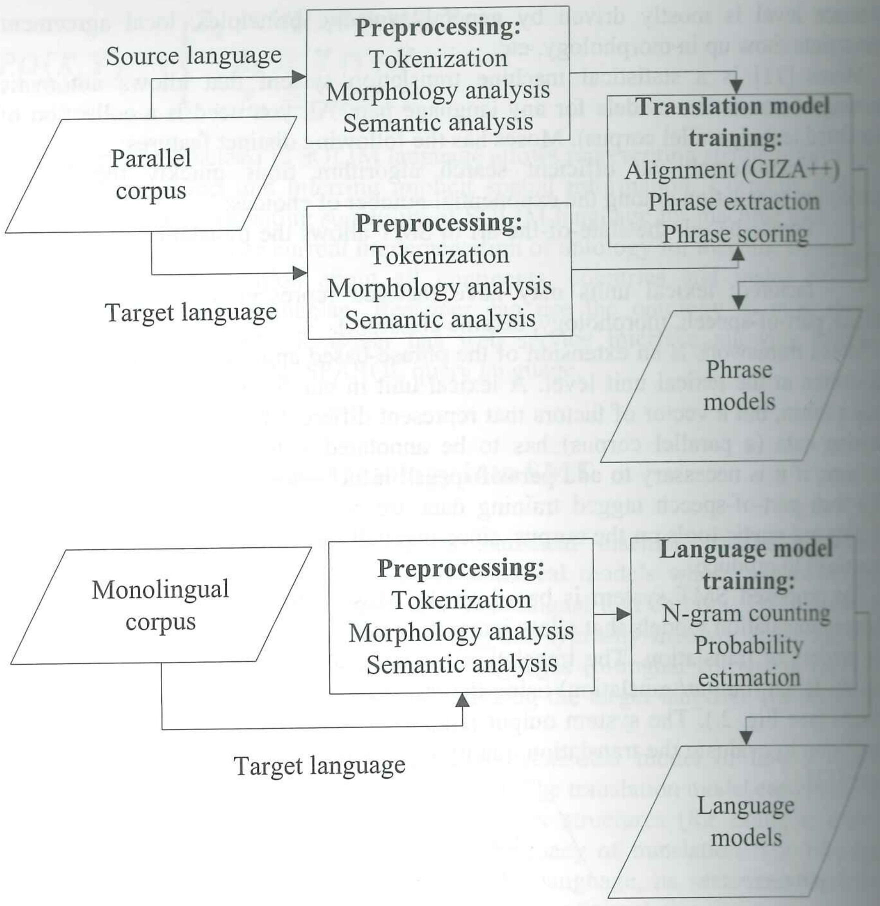

Fig. 3. Training process.

Text pre-processing is used both for training and translation tasks. During training, both parallel and monolingual training corpora are preprocessed. During translation, the input text is pre-processed in the same way as for training.

Pre-processing includes the following steps:

- Sentence splitting;
- Tokenization (splitting of text into the smallest translation entities, e.g. words, multiword entities, punctuation symbols);
- Case processing (e.g., lowercasing the first word of sentence);
- Morphologic analysis (optionally);
- Semantic analysis (using the semantic reasoner to augment input tokens with semantic factors).

The inclusion of morphological and semantic analysis depends on the characteristics of the source and target languages and of the translation task itself. Separate pre-processor must be implemented for the source and target languages.

The post-processing works with the output of Moses decoder (translation target language) to convert it to orthographically correct text. It involves the following tasks:

- De-tokenization (gluing of tokens to comply with orthography, e.g. no whitespace before comma);
- Case processing (e.g., capitalizing the first word of each sentence);
- Sentence gluing.

Semantic data has been integrated into the SMT system as additional factors on both source and target language side. By factoring semantic information into the source language, the translation accuracy is improved by resolving semantic ambiguities in the source language. Similarly, by factoring semantic information into the target language, the translation fluency is improved by resolving semantic ambiguities in the target language. The use of factors implies a certain tag set for each factor. The input text will be analyzed and tagged with the information from a knowledge base inferred by the semantic reasoner. Semantic factors, or tags, will be produced for each relevant token in the input text. One or several tags can be used, depending on the particular ambiguities that are targeted. For example, an English place name *Georgia* is ambiguous. It might refer to the US state or the country in Caucasian region. Consider the following ambiguous sentences in English:

1. *The president of Georgia visited Lithuania yesterday.*
2. *There are Lithuanians living in Georgia and other states.*
3. *A recent court ruling gives Georgia, Alabama and Florida three years to resolve their conflict over use of the water from Lake Lanier.*
4. *Experts have failed to travel to Georgia at the Tbilisi international airport.*
5. *Azerbaijan has bad relations with Armenia, so it cannot afford to have such relations with Georgia.*
6. *US president visited Georgia last month.*

It is quite obvious that in examples 1, 4 and 5 *Georgia* stands for the country, but in examples 2 and 3 it stands for the US state. Example 6 is very ambiguous and it is not possible to determine what is meant by *Georgia* without a wider context.

Using typical SMT training methods without tags to train translation models on big amount of parallel sentences results in *Georgia* translated by several different Latvian translation equivalents with different probabilities. The results are as follows:

Table 1. Simple translation probabilities of word *Georgia*.

English	Latvian	probability
Georgia	Gruzija	0.45
Georgia	Džordžija	0.42
Georgia	Grūzija	0.08
Georgia	…	
Georgia	Tibilisi	0.001

So, there are 2 translations with a quite high probability, these both are valid translations. The first is a translation of the country *Georgia*, the second is a translation of the state *Georgia*. Other translations are with very low probability and come from errors in training data.

If probabilities are calculated for phrase translations some more hints could be obtained:

Table 2. Simple translation probabilities of phrases containing word *Georgia*.

English	Latvian	probability
President of Georgia	Gruzijas presidents	0.9
Georgia Bulldogs	Džordžijas Buldogi	0.8
	…	
President of Georgia	Džordžijas presidents	0.000003

This means that phrase *president of Georgia* will be almost sure translated as *Gruzijas prezidents* and not *Džordžijas prezidents* and *Georgia Bulldogs* will be surely translated as *Džordžijas Buldogi* not *Gruzijas Buldogi*. It can be seen that neighboring lexical units in the context can help to disambiguate phrase translation. Tags can be also assigned to ambiguous lexemes to be disambiguated. Typically part of speech tags are used in factored SMT. For example, if translation probabilities of an English lexeme fish are calculated, the following results might be obtained:

Table 3. Simple translation probabilities of words.

English	Latvian	probability
fish	zivs	0.8
fish	zvejot	0.1
	…	
fish	tips	0.01

But if lexemes are tagged with part of speech information then the following results might be obtained:

Table 4. Factored translation probabilities of words.

English	Latvian	probability
fish|N	zivs	0.9
fish|V	zvejot	0.5
fish|V	makšķerēt	0.4
	…	
fish|N	tips	0.01

The second table is more precise and more useful in translation, because it gives context dependent and more reliable probabilities.

Other types of tags can be also used to improve an SMT system. There are several ways how lexical units can be tagged with the information obtained from the spatial ontology. For example, if there is a sentence containing lexical units *X* and *Y* and the information about a relation *Z* between *X* and *Y* can be obtained from the knowledge base, then the lexical unit *X* can be tagged with tag *X(Z.Y)*. For example, if there is a sentence

Azerbaijan has bad relations with Armenia so it cannot afford to have such relations with Georgia

a lexeme *Georgia* can be tagged *EC.Armenia* because it can be derived from the ontology that there is a relation *EC* between these two lexemes *Georgia* and *Armenia*. If there is a sentence

Experts have failed to travel to Georgia at the Tbilisi international airport

the lexeme *Georgia* can be tagged *NTPPi.Tbilisi* because it can be derived from the ontology that there is a relation *NTPPi* between the two lexemes *Georgia* and *Tbilisi*.

SMT training data annotated with such tags will lead to better translation probabilities which are context dependent. The following results could be obtained:

Table 5. Factored translation probabilities of word *Georgia*.

English	Latvian	probability
Georgia|EC.Armenia	Gruzija	0.7
Georgia|NTPPi.Tbilisi	Gruzija	0.8
Georgia|TPP.Caucasus	Grūzija	0.08
Georgia|EC.Florida	Džordžija	0.8
Georgia|NTPP.US	Džordžija	0.5
Georgia|NTPP.US	Gruzija	0.4

The first line in the Table 5 means that a probability of a translation equivalent for the lexeme *Georgia* is *Gruzija* is 0.7, given that there is also a lexeme *Armenia* in the same sentence and there is *EC* relation between the two lexemes *Georgia* and *Armenian* in the ontology.

The described approach is still probabilistic and not knowledge-based. As it can be seen from the Table 5, the fact that there is an *NTPP* relation between the two lexemes *Georgia* and *US* which are in the same sentence does not mean that *Georgia* must be translated as *Džordžija*. *US* might be equally often used together with state *Georgia* and country *Georgia*.

Annotation must be performed both for training corpus and for translation input. During the translation process the factors provide additional information to the decoder, which helps to choose the appropriate translation equivalent. For annotating the data it is necessary to derive the information – the relevant tags, form the spatial ontology.

For this, firstly, the ontology must be able to be queried for all the ontological entities – a list of them: e.g. *GetAllSpatialObjects*: a query to the spatial ontology. This method returns a list of names.

Secondly, after all the relevant lexical units are identified and marked in the texts, it is necessary to derive further information from the spatial ontology (relations between identified lexical units), e.g. *GetSpatialRelations(A, B)*: a query to the spatial ontology to get knowledge from it. This method have 2 parameters (names of spatial objects from spatial ontology) and returns a list of relations inferred by the spatial reasoned (it does not state anything about the nature of relations that are false or are unknown). For example, it is important to know, that *Tbilisi* is a city and it is a city in *Georgia (Georgia NTPPi Tbilisi)*. This information can influence translation results and will be used in the training process. Another query *GetSpatialRelations(Georgia,Armenia)* returns the relation *EC* only if there is enough information in the ontology to infer this relation (using open world assumption), whereas the query *GetSpatialRelations(Georgia,Latvia)* returns the relation *DC* if this relation can be inferred (using open world assumption) and *GetSpatialRelations(Georgia,Lativa)* returns nothing if there is no enough information in the ontology to infer *DC* relation.

Implemented methods for ontology querying are used to annotate both SMT training data and MT input text with factors containing spatial knowledge.

4 Current Results and Future Work

This paper presents ongoing research. SOLIM spatial ontology language used in this project is specified and implemented. But it is in evaluation stage now; it might be altered depending on evaluation results. Use of SOLIM language in machine translation system is one of its' evaluation scenarios, other evaluation scenario is related to image search. Spatial ontology containing information about all continents, countries, USA states, world's major cities and all populated places in Latvia and Lithuania has been implemented and tested. There is a reasoner created for SOLIM ontology, and it is available throw a web service interface using SPARQL query language. SMT training and translation processes have been adapted so that SMT system can use reasoner to annotate SMT training data and to preprocess input sentences. Test SMT system has been trained, and it is ready for evaluation and elaboration.

Currently we have SMT system which is enriched with semantic data coming from a spatial ontology. Spatial knowledge definitely gives an improvement of SMT system. But this improvement has to be formally evaluated using MT quality evaluation metrics, such as BLEU score [12] or HTER [13]. Current implementation of ontology integration uses all spatial relation which we can get from the reasoner, but it is obvious that different relations have different impact on translation quality. Impact of each spatial relation needs to be evaluated and optimal combination needs to be found. Spatial information is just one type of semantic knowledge which can be added to SMT system. Enriching SMT with other types of semantic knowledge coming from other types of ontologies is also a perspective research direction.

Acknowledgments. The research within the project SOLIM leading to these results has received funding from the European Union EUREKA's Eurostars Programme, grant agreement no E! 4365. I would like to thank Karlis Goba, Tatiana Gornostay, Thomas Dohmen and Mark Vreijling for contributing to parts of this paper.

References

1. Brown, P., Della Pietra, S., Della Pietra, V., Mercer, R.: The mathematics of statistical machine translation: Parameter estimation. Computational Linguistics, 19(2):263-311 (1993).
2. Koehn, P., Och, F. J., Marcu, D.: Statistical phrase based translation. In Proceedings of the Joint Conference on Human Language Technologies and the Annual Meeting of the North American Chapter of the Association of Computational Linguistics (HLT/NAACL) (2003)
3. Koehn, P. and Hoang, H.: Factored Translation Models. In Proceedings of EMNLP'07 (2007).
4. Chiang, D.: Hierarchical Phrase-Based Translation. In Computational Linguistics 33(2), pp. 201-228 (2007)

5. Marcu, D., Wang, W., Echihabi, A., and Knight, K.: SPMT: statistical machine translation with syntactified target language phrases. In Proc. of the 2006 Conference on Empirical Methods in Natural Language Processing (Sydney, Australia, July 22 - 23, 2006). ACL Workshops. Association for Computational Linguistics, Morristown, NJ, pp. 44-52. (2006)
6. Li, Z., Callison-Burch, C., Dyer, C., Ganitkevitch, J., Khudanpur, S., Schwartz, L., Thornton, W., Weese, J., Zaidan, O.: Joshua: An Open Source Toolkit for Parsing-based Machine Translation. In Proceedings of the Workshop on Statistical Machine Translation (WMT09), (2009)
7. Quirk, C., Menezes, A., Cherry, C.: Dependency Treelet Translation: Syntactically Informed Phrasal SMT, In Proc.of ACL 2005 (2005)
8. Randell, D. A., Cui, Z. and Cohn, A. G.: A spatial logic based on regions and connection, Proc. 3rd Int. Conf. on Knowledge Representation and Reasoning, pp. 165–176, Morgan Kaufmann, San Mateo (1992)
9. Egenhofer, M. J. and Franzosa, R.: Point-Set Topological Spatial Relations. International Journal of Geographical Information Systems, Vol. 5(2), pp. 161--174 (1991)
10. Katz, Y. and Cuenca Grau, B.: Representing Qualitative Spatial Information in OWL-DL. In Proc. of 1st OWL: Experiences and Directions Workshop (OWLED2005), Galway, Ireland (2005)
11. Koehn, P., Federico, M., Cowan, B., Zens, R., Duer, C., Bojar, O., Constantin, A., Herbst, E.: Moses: Open Source Toolkit for Statistical Machine Translation. In Proceedings of the ACL 2007 Demo and Poster Sessions, pp. 177-180, Prague (2007)
12. Papineni, K., Roukos, S., Ward, T., Zhu, W.: BLEU: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting of the Association of Computational Linguistics (ACL) (2002)
13. Snover, M., Madnani, N., Dorr, B.J., Schwartz, R.: Fluency, adequacy, or HTER?: exploring different human judgments with a tunable MT metric, Proceedings of the Fourth Workshop on Statistical Machine Translation, Athens, Greece (2009)