

# Model Driven Specification of Ontology Translations

Fernando Silva Parreiras<sup>1</sup>, Steffen Staab<sup>1</sup>, Simon Schenk<sup>1</sup>, and Andreas Winter<sup>2</sup>

<sup>1</sup> ISWeb — Information Systems and Semantic Web,  
Institute for Computer Science, University of Koblenz-Landau  
Universitaetsstrasse 1, Koblenz 56070, Germany  
{parreiras, staab, sschenk}@uni-koblenz.de

<sup>2</sup> Institute for Computer Science, Johannes-Gutenberg-University Mainz  
Staudingerweg 9, Mainz 55128, Germany  
winter@uni-mainz.de

**Abstract.** The alignment of different ontologies requires the specification, representation and execution of translation rules. The rules need to integrate translations at the lexical, the syntactic and the semantic layer requiring semantic reasoning as well as low-level specification of ad-hoc conversions of data. Existing formalisms for representing translation rules cannot cover the representation needs of these three layers in one model. We propose a metamodel-based representation of ontology alignments that integrate semantic translations using description logics and lower level translation specifications into one model of representation for ontology alignments.

## 1 Introduction

The reconciliation of data and concepts from different ontologies and data repositories in the Semantic Web requires the discovery, the representation and the execution of ontology translation rules. Though most research attention is now devoted to the discovery of alignments between ontologies, a shallow inspection of ontology alignment challenges already reveals that there does not exist *one* easily accessible way of representing such alignments as translation rules [1].

The reason is that alignments must address ontology translation problems at different layers [2] [1]:

1. At the *lexical layer* it is necessary to arrange character sets, handling token transformations.
2. At the *syntactic layer* it is necessary to shape language statements according to the appropriate ontology language grammar.
3. At the *semantic layer* it is necessary to reason over existing ontological specifications and data in both the source and the target ontologies.

For addressing ontology translation problems at the semantic layer, existing frameworks provide reasoning in one or several logical paradigms, such as

description logics [3] [4] or logic programming [5] [6] [7]. For addressing ontology translation problems at lexical and syntactic layers, alignment frameworks take advantage of platform-specific implementations, sometimes abstracted into translation patterns [8] [9] or into logical built-ins [7].

Such hybrid approaches, however, easily fail to provide clarity and accessibility to the modelers who need to see and understand translation problems at semantic, lexical and syntactic layers. Indeed, modelers need to manage different languages: (1) an ontology translation language to specify translation rules and (2) a programming language to specify built-ins, when the ontology translation language does not provide constructs to completely specify a given translation rule. This intricate and disintegrated manner draws their attention away from the alignment task proper down into diverging technical details of the translation model.

Filling the gap in ontology translation domain between ontology mapping languages and general purpose programming languages helps to improve productivity, since modelers will not have to be aware of platform-specific details and will be able to exchange translation models even when they use different ontology translation platforms. Moreover, maintenance and traceability would be facilitated because mapping knowledge is not longer embedded in source code of programming languages anymore.

We propose an platform independent approach for ontology translation based on model-driven engineering (MDE) of ontology alignments. The framework includes a language to specify ontology translations, the Model-Based Ontology Translation Language (MBOTL). In order to reconcile *semantic* reasoning with idiosyncratic *lexical* and *syntactic* translations, we integrate these three different translation problems into a representation based on a joint metamodel. The joint metamodel comprises, among others, the OWL-DL metamodel and the OCL metamodel to support specification, representation and execution of ontology translations.

The paper is organized as follows: The running example and the requirements for ontology translation approaches are explained in Section 2. Our solution is described in Section 3, followed by examples in Section 4. In Section 5 we discuss the requirements evaluation and in Section 6 we present related work. The conclusion, Section 7, finishes the paper with an outlook to future work.

## 2 Running Example and Requirements

We consider two ontologies of bibliographic references from the test library of the Ontology Alignment Evaluation Initiative (OAEI) [1] to demonstrate the solution presented in this paper: the reference ontology (#101) and the Karlsruhe ontology (#303). Canonical mappings covered by examples in this paper and snippets of the source and target ontologies using the Manchester OWL Syntax are shown in Fig. 1. Please refer to OAEI for complete ontologies.

By examining the mapping between ontology #101 and ontology #303, it becomes clear that translations are required in order to completely realize the

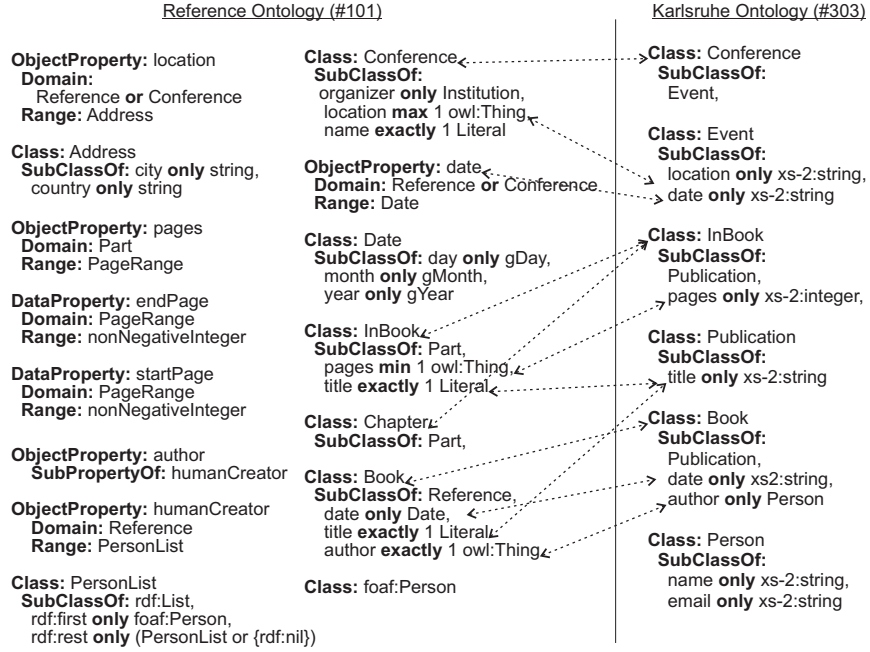


Fig. 1. Ontology mapping challenge for the running example.

mapping. Individuals of the classes **Chapter** and **InBook** in ontology #101 are translated into individuals of the class **InBook** in the ontology #303. Values of the object property **month** having a Gregorian month, e.g., ‘--01’, are translated into the equivalent unabbreviated form, e.g., ‘January’. Values of the data property **pages** in ontology #303 can be calculated by subtracting the value of the data property **initialPage** from the value of the property **endPage** in ontology #101.

We define the translation rules explained above by the following logical rules. All variables are treated as universally quantified and prefixed with a question mark. Let *builtin: notShortened* be a built-in function that returns the unabbreviated month, *builtin: toUpper* be a built-in function to capitalize strings, *builtin: -* be a subtractor function, *s* be the namespace prefix of the source ontology #101, and *t* be the namespace prefix of the target ontology #303, the translation rules can be written as follows:

$$\begin{aligned}
& t: InBook(?x) \wedge t: month(?x, ?m) \wedge t: title(?x, ?n) \wedge t: pages(?x, ?p) \leftarrow \\
& \quad (s: InBook(?x) \vee s: Chapter(?x)) \wedge s: month(?x, ?y) \wedge \\
& \quad builtin: notShortened(?y, ?m) \wedge s: title(?x, ?z) \wedge builtin: toUpper(?z, ?n) \wedge \\
& \quad s: pages(?x, ?w) \wedge s: startPage(?w, ?a) \wedge s: endPage(?w, ?e) \wedge \\
& \quad \quad \quad builtin: -(?e, ?a, ?p). (1)
\end{aligned}$$

The translation rule of authors is not trivial as well. While in ontology #101 the authors are collected by recursively matching the property `first` of the class `PersonList`, in ontology #303 it is a matter of cardinality of the object property `author`. Let `list:contains` be the built-in able to filter a list structure into object properties, the referred rule can be written as follows:

$$\begin{aligned} & t: Book(?x) \wedge t: author(?x, ?u) \leftarrow \\ & s: Book(?x) \wedge s: author(?x, ?y) \wedge list: contains(?y, ?u). \end{aligned} \quad (2)$$

However, built-ins are black boxes that conceal knowledge about algorithms, compromising traceability and maintenance. Therefore, an approach able to specify rules and built-ins without code specifics is required.

From inspecting these examples, we illustrate requirements for a platform independent ontology translation approach addressing translation problems at the following ontology translation layers proposed by Corcho and Gómez-Pérez [2] based on Euzenat [1]: the lexical layer, the syntactic layer, the semantic layer and the pragmatic layer. Since the pragmatic layer addresses the meaning of representation in context, it is similar to the semantic layer from the point of translation decisions. In this paper, we refer to both layers as semantic layer.

1. The lexical layer deals with distinguishing character arrangements, including:
  - (a) *Transformations of element identifiers*. They are required when different principles are applied to name objects, for example, when transforming the value of the data property `title` into capital letters (Rule 1).
  - (b) *Transformations of values*. They are necessary when source and target ontologies use different date formats, for example transforming a Gregorian month into an unabbreviated form (Rule 1).
2. The syntactic layer covers the anatomy of the ontology elements according to a defined grammar. The syntactic layer embraces:
  - (a) *Transformations of ontology element definitions*. They are needed when the syntax of source and target ontologies are different, e.g., when transforming from OWL 1.0 RDF syntax<sup>3</sup> into OWL 1.0 XML syntax<sup>4</sup>.
  - (b) *Transformations of datatypes*. They involve the conversion of primitive datatypes, e.g., converting string datatype to date datatype.
3. The semantic layer comprises transformations dealing with the denotation of concepts. We consider two different aspects:
  - (a) *Inferred knowledge*. Reasoning services are applied to deduce new knowledge, for example, inferring properties from class restrictions.
  - (b) *Transformations of concepts*. It takes place when translating ontology elements using the same formalism, e.g, translating a concept from Karlsruhe's OWL ontology for bibliographic references into one or more concepts in the INRIA's OWL ontology.

<sup>3</sup> <http://www.w3.org/TR/rdf-syntax-grammar/>

<sup>4</sup> <http://www.w3.org/TR/owl-xmlsyntax/>

The translation problems are classified in non-strict layers, e.g., one rule commonly addresses more than one translation problem. For example, in Rule 2, the built-in `toUpper` solves a translation problem at the lexical layer, the translation of months happens at the syntactical layer and is achieved by the built-in `notShortened` and, finally, the translation of the union of individuals of the classes `Chapter` and `InBook` in ontology #101 into individuals of the class `InBook` in ontology #303 appears at the semantic layer.

An orthogonal classification of ontology translation problems is given by Dou *et al.* [6]. From their point of view, ontology translation problems comprise dataset translation, ontology-extension generation and querying through different ontologies. This paper concentrates on dataset translation, i.e., translation of instances, leaving the model driven engineering of the remaining problems for future work.

### 3 A Model Driven Framework for Ontology Translations

The proposed ontology translation approach relies on advances in Model Driven Engineering (MDE) with support for Description Logic reasoning services [10] [11]. We define here the Model-Based Ontology Translation Language (MBOTL) comprising (1) a textual concrete syntax used to write translation rules, (2) an integrated metamodel as abstract syntax to represent the translation rules as models, (3) an extensible model library to provide built-in constructs, (4) model transformations yielding translational semantics and (5) a pilot implementation with model transformations to the target framework implementing ontology translation, in this case SPARQL and Java. Please, refer to the project web site [12] for complete specifications of these artifacts.

#### 3.1 Concrete Syntax

While visual notations are effective in communicating models, textual notations are preferable to express more complex structures. The following subsections present the anatomy of the translations rules, alluding to the requirements presented in Section 2.

**Dealing with Translation Problems at Semantic Layer** In order to extract information from the source ontology, we need a query language able to determine which datasets are to be translated. We use OCL expressions to formulate queries. Indeed, OCL has been used in MDE for specifying constraints and queries that are side effect free operations. As OCL is originally designed to be used with UML or MOF, we have extended OCL to be used with OWL [11], i.e., to support reasoning operations.

Ontology translation problems at the semantic layer are treated by querying individuals of the source ontology using OCL queries and matching target individuals. Queries are part of the input pattern in a transformation rule, that has an output pattern as well with variables binding the elements. Variables are

declared and used as in classical programming. These assumptions have been used by model transformation languages like OMG MOF Query/View/Transformation (QVT) and the Atlas Transformation Language (ATL) [13]. We base MBOTL upon the ATL concrete syntax to specify ontology translations because it is simpler and more intuitive.

The example depicted in Fig. 2 illustrates the concrete syntax. A rule `Conference2Conference` is defined for translating individuals of the class `Conference` in ontology #101 into individuals of the class `Conference` in ontology #303.

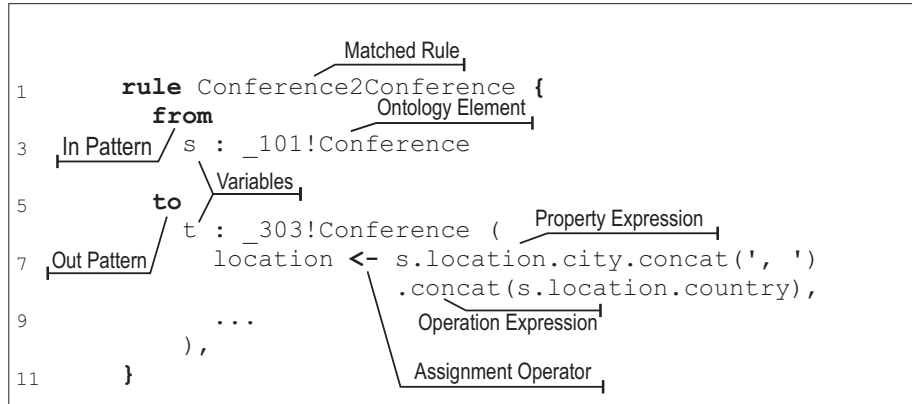


Fig. 2. Example of a Translation Rule

In OCL, a dot-notation is used to navigate through properties. In the scope of our extension of OCL, a property can be an OWL data property, an OWL object property, a predefined operation or a helper. A helper is a user defined side effect free query operation belonging to a defined class in one of the given ontologies.

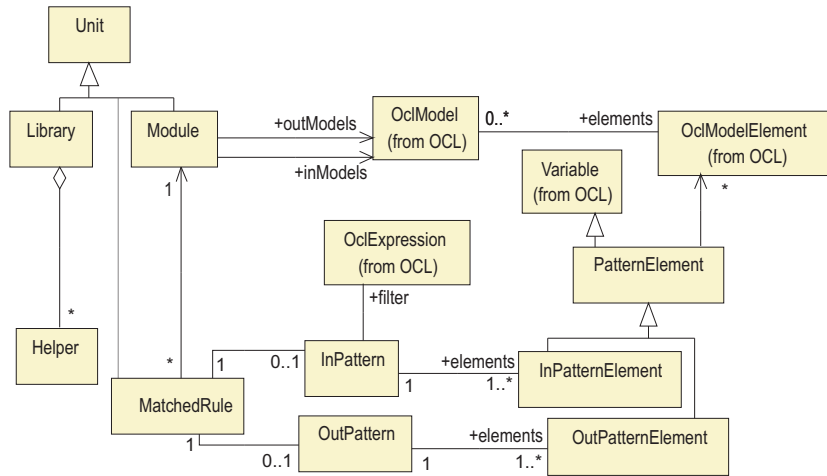
For example, in the expression `s.location`, `s` is a reference to an individual of the class `Conference` with `location` resulting in a value of the class `Address`. The navigation can also end with an operation evaluation, as depicted in Fig. 2, where the operation `concat` is used to concatenate the properties `city` and `country`.

**Addressing Translation Problems at Lexical and Syntactic Layers** Ontology translation problems at lexical and syntactic layers are supported by means of employing operations or helpers. For example, for the type `string`, the operation `toUpper()` returning an string object with capital letters is available. Thus, the evaluation of `s.title.toUpper()` capitalizes the value of the property `title`.

The operation `toUpper()` is an example of predefined operation. The set of predefined operations is available in the OCL library (M1 layer). These operations are applicable to any type in OCL. Additionally, it is possible to specify *ad hoc* operations, the so-called helpers.

### 3.2 Metamodels

The textual concrete syntax for ontology translation specification presented in the previous section has an integrated metamodel as equivalent abstract syntax. The integrated metamodel consists of the following metamodels: MOF metamodel, OCL metamodel, OWL metamodel [14], and part of the ATL metamodel [13]. As a matter of space, we do not present the complete metamodels in this paper, but noteworthy fragments. A detailed version is available online [12].

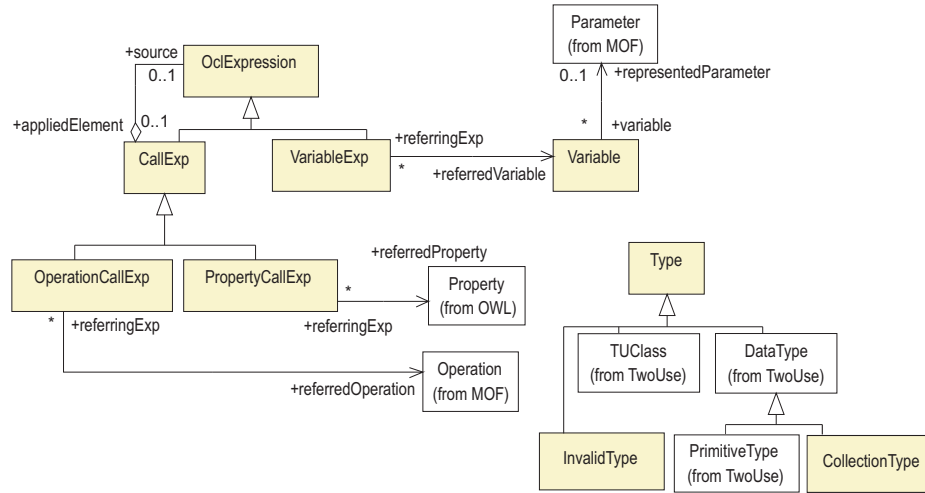


**Fig. 3.** Fragment of the ATL Metamodel.

The translation metamodel (Fig. 3) allows for describing translations between two ontologies by means of a model. A translation is characterized as a `Module` relating source ontologies (`inModels`) and target ontologies (`outModels`). A `MatchedRule` is a specific translation rule that has a pattern for the input model (`inPattern`) and a pattern for the output model (`outPattern`). The `InPattern` has one or more elements that are OCL variables (`Variable`). Variables are bound to model elements (`OclModelElement`). The `InPattern` has an `OclExpression` acting as query to refine individuals of the `OclModelElement`.

Since each expression in OCL has a type, we need a type metamodel (Fig. 4). The expression evaluation produces a value of type of the expression. The type `TUClass` is the particular composition of the OWL class with the MOF class. This composition allows for applying side effect free operations into individuals of OWL classes, e.g., reasoning operations.

Figure 4 depicts additionally another part of the integrated metamodel, namely the package `Expressions` of the extended OCL metamodel. The class `OclExpression` enables MBOTL to define the abstract syntax for OCL expressions. The integration with the OWL metamodel is accomplished by expressions of type `PropertyCallExp`. Such expression allows for navigating through OWL properties, as explained in Sect. 3.1.



**Fig. 4.** Snippet of the package `Type` and package `Expressions` of the OCL metamodel

The operation call expressions (`OperationCallExp`) support the declaration of built-in operations and helpers. An operation call expression evaluates to the result of a class operation, providing that such operation is side effect free. This resource is particularly relevant in the scope of ontology translation, i.e., it enables queries to invoke built-in reasoning operations or helpers.

### 3.3 Model Libraries

The model libraries define a number of datatypes, class identifiers and operations that must be included in the implementation of MBOTL. These constructs are instances of an abstract syntax class. The foundation library exists at the M1 level, where the abstract syntax (metamodel) exists at M2 level. The foundation library is composed of the XML Schema Datatypes library, the RDF library, the OWL library and the OCL library.

Examples of M1 objects of the XML Schema datatypes library are the datatypes `gDay`, `gMonth` and `gYear`, having the M2 class `RDFS::RDFSDataType` as metaclass. In the RDF library, for example, the M1 object `nil` has the M2 class `RDFS::RDFList` as metaclass. In the OWL library, interesting M1 objects

are **Thing** and **Nothing**, both having the M2 class `OWL::OWLClass` as meta-class. These three libraries are based on the foundation library for RDF and OWL described in the ODM specification [14].

An example of M1 object of the extended OCL library is the construct `oclAny`. All types inherit the properties and operations of `oclAny`, except collection types. This invariant allows for attributing predefined operations to classes. The OCL library is based on the standard OMG OCL library.

### 3.4 Semantics

The semantics of MBOTL is defined by the semantics of the languages comprising the integrated metamodel (Sect. 3.2).

MBOTL is translated into a target language (SPARQL and Java). Regarding the target languages, the semantics of SPARQL is described by algebraic semantics whereas the semantics of Java can be defined by providing an Abstract State Machine [15]. More specifically, the SPARQL basic graph pattern is described according to an entailment regime. Indeed, SPARQL-DL [16] provides an entailment regime for OWL-DL.

### 3.5 Ontology Translation Process

In order to guide the user from the ontology translation specification until the running code, the ontology translation process covers the following steps:

1. *Specification of Ontology Translation.* The ontology translation rules and helpers are specified by the user using MBOTL.
2. *Specification of Model Transformations.* In order to have a running implementation of ontology translation, the ontology translation specification model is transformed into models for a given platform. The model transformation specification mapping the MBOTL model onto platform specific models must be specified here. Our framework provides model transformations from MBOTL into SPARQL and Java as target platforms. Notice that other target platforms like F-Logic (Ontobroker) can be considered.
3. *Transformation into Target Platform.* Three transformations take place at this step. Firstly, the ontology translation specification in the concrete syntax (MBOTL file) is injected into a model conforming with the integrated metamodel, i.e, the ontology translation specification model. The second transformation is responsible for generating models according to the target metamodels, e.g., SPARQL and Java metamodels. Thirdly, SPARQL queries in the SPARQL concrete syntax and Java code are extracted from the SPARQL and Java MOF-based models.

### 3.6 Implementation

The implementation comprises (1) the environment to specify ontology translations and (2) transformations into ontology translation engines in order to realize ontology translation.

We have an implementation covering the MDE process for parts of MBOTL and we are currently working towards a comprehensive solution. Our implementation uses the Generative Modeling Technologies (GMT) project [17] under the Eclipse Modeling Framework. The Textual Concrete Syntax component (TCS) [18] of the Eclipse GMT is used to specify the concrete syntax used to write dataset translations (1). Furthermore, such component allows for automatically translating the specification into a model conforming with the proposed integrated metamodel, i.e., the ontology translation specification model.

Taking the ontology translation specification model as source model, we use the Atlas Transformation Language [13] framework including Textual Concrete Syntax (TCS) to define model transformations into models for an ontology translation platform (2). We are currently using SPARQL and Java as target languages and the Jena framework as ontology translation solution. The Jena framework includes an API for OWL ontologies and reasoners, as well as a SPARQL engine.

Elements of the ontology translation specification model concerning translation problems at the semantic layer are transformed by ATL into SPARQL CONSTRUCT queries. The SPARQL engine can be extended using custom SPARQL filter functions — as foreseen as an extension hook in the SPARQL standard, but also using so called *predicate functions*. Predicate functions are not matched against the knowledge base like normal RDF predicates, but evaluated in Java code. Filter and predicate functions are used to handle translation problems at the lexical and syntactic layer. These functions are defined in the ontology translation specification model and have the Java code automatically generated by the ATL transformation.

The next section illustrates our approach by addressing the translation problems presented in Section 4, specifying the translation rules and transforming the ontology translation specification into SPARQL and Java code.

## 4 Application

This section presents rules integrating translations problems at semantic, syntactic and lexical layers, according the problems presented in Section 2. For further details and examples please refer to the Technical Report [12].

The classes `Chapter` and `InBook` in ontology #101 are translated into the class `InBook` in the ontology #303. The translation rule uses a helper to transform a Gregorian month, e.g., ‘--01’, into its equivalent unabbreviated form, e.g., ‘January’. This helper is applicable only to the `gMonth` datatype. Using MBOTL, we can specify both the rule and the helper — and hence lexical, syntactical and semantical translations — using an integrated framework. The helper is shown on top of listing 1.1, followed by the translation rule.

---

### Listing 1.1. Semantic, syntactic and lexical translations with MBOTL

---

```
1 helper context _101!gMonth
   def: notShortened() : String =
```

```

Sequence{ 'January', 'February', 'March' }->at (
  Sequence{ '--01', '--02', '--03' }->indexOf(self.toString()))
5
rule ChapterInBook2Inbook {
  from
    s : _101!Part (s.owlIsInstanceOf(Chapter) or
                  s.owlIsInstanceOf(Inbook))
10  to
    t : _303!Inbook (
      title <- s.title.toUpper(),
      pages <- s.pages.endPage - s.pages.startPage,
      month <- s.date.month.notShortened(),
15  )
}

```

---

After we have been able to specify all aspects of the mapping in MBOTL, it is translated into suitable languages for execution. Our implementation uses SPARQL queries for semantic mappings and Java code for syntactic translations.

As we can see from the examples, helpers are used for lexical and syntactical translations and semantic translations.

## 5 Requirements Evaluation and Discussion

In response to the requirements deduced in Sect. 2, Table 1 shows use cases according to each requirement and where to find the corresponding examples in this paper.

**Table 1.** Satisfying ontology translation requirements

Requirement (Sect. 2)	Use Case	Implementation
1.(a)	converting to capital letters	Listing 1.1, Line 12
1.(b)	converting date formats	Listing 1.1, Line 14
2.(b)	converting <code>gMonth</code> to <code>String</code>	Listing 1.1, Line 14
3.(a)(b)	Union of <code>Chapter</code> and <code>InBook</code>	Listing 1.1, Line 8-9

Translation problems of lexical nature, like converting a string to an upper case string, are managed by using predefined OCL operations applied to specific types of objects, in this example a string type. It is also possible to write functions, i.e., helpers, to perform *ad hoc* operations. For example, the helper `notShortened` (Listing 1.1) allows for converting date formats, i.e., replacing a value of `gMonth` type to the unabbreviated form.

Translation problems inherent in the syntactic layer are handled distinctly. While datatype conversions are achieved by invoking predefined operations, like `toString()` (Listing 1.1), the translation from OWL RDF/XML to OWL XML

can be accomplished by injectors and extractors to serialize the models (not shown in this paper).

Translation problems at the semantic layer, regarding datasets of ontologies with different vocabularies but the same formalism is demonstrated by the running example. In Listing 1.1, the individuals of the class `Chapter` in ontology #101 and the individuals of the class `InBook` are translated into individuals of the class `InBook` in ontology #303.

*Limitations* . Our approach has some restrictions reflected by the ATL meta-model. With ATL, it is possible to realize only unidirectional translations. A bidirectional translation must be accomplished by two unidirectional translations.

Moreover, at the current state of development, it is not possible to validate or to reason over translation models. In other words, it is not possible to test the translation model without transforming it into the target platform (SPARQL and Java).

## 6 Related Works

Since a lot of work has been done in the field of ontology alignment, we group works according to semantic, syntactic and lexical layers.

Among works covering lexical and syntactic translations, Model transformation languages like OMG QVT and ATL [13] allow for defining how to transform MOF-based models using declarative and imperative constructs. Nevertheless, their semantics does not support reasoning over OWL ontologies. Our contribution extends ATL by integrating with the OWL metamodel and rewriting OCL semantics to support querying OWL ontologies.

The work of Atzeni *et al.* [19] is based on a metamodel approach with models described in terms of the constructs they involve, taken from a given set of predefined ones. However, the work is in the scope of databases and does not support reasoning at the semantic layer.

Among works covering semantic reasoning capabilities, C-OWL [3] and the ontology mapping system proposed by Haase and Motik [4] are formal solutions for ontology mapping with DL expressiveness. The mappings are based on subsumption relationships of concepts between ontologies. Notwithstanding, the usage of built-ins to express lexical and syntactic translation problems is not possible. A metamodeling-based approach of Haase and Motik [4] is provided by Brockmans *et al.* [20]. Although the usage of built-ins in mapping rules is allowed, the latter approach does not provide means to specify built-ins without recourse to programming languages, whereas MBOTL allows for specifying *ad hoc* functions by means of helpers.

Among works covering lexical, syntactic and semantic translations, MAFRA [8] and RDFT [9] are frameworks enabling dataset translations. Nonetheless, both are based on RDF Schema and neither they provide the expressiveness of OWL-DL nor support reasoning capabilities of DL inference engines.

OntoMorph [5] and the framework proposed by Dou [6] for ontology translation rely on First Order Logic (FOL) expressiveness to specify translation rules. Our approach counts on the decidable subset of FOL, i.e., Description Logics with complete and sound automated reasoning services for addressing semantic translation problems. Moreover, while the first solution relies on PowerLoom and the latter on Web-PDDL, we propose a platform independent model-based translation language, flexible enough to be used with different knowledge representation system.

OntoMap [7] is a mapping solution allowing for visual specification of mappings, with a limited number of translation functions. Snoogle [21] is an ontology translation tool that enables the use of SWRL rules to express translations and alignments between geospatial ontologies. While in both approaches it is possible to use custom plug-ins, the user has to write functions using Java and the Jena framework. In contrast, our approach allows for specifying mapping rules and functions in a platform independent and integrated way.

Corcho and Gómez-Pérez [22] propose ODEDialect, a set of declarative languages to specify ontology translations. However, it is platform specific approach based on Java that exposes users to complexity of programming languages, whereas MBOTL allows modelers to concentrate on business logics instead.

## 7 Conclusion

This paper presents a solution for ontology translation specification that aims at being more expressive than ontology mapping languages and less complex and fine-grained than programming languages. The solution is comprised of a concrete syntax, an integration metamodel covering OWL, MOF, OCL and ATL metamodels and model transformations from MBOTL into SPARQL and Java. We validate our solution against canonical ontology translation problems grouped into three layers: lexical, syntactic and semantic.

*Future Work.* Future areas of investigation involve different ontology translation problems like query translation and ontology-extension generation. The application of the proposed solution in networked environments is of particular interest for the field of distributed ontologies as well as the integration with the ontology mapping metamodel. Therefore, we plan to integrate the Eclipse Plug-ins into the Neon toolkit<sup>5</sup>.

## References

1. Euzenat, J., Shvaiko, P.: *Ontology Matching*. Springer (2007)
2. Corcho, Ó., Gómez-Pérez, A.: A layered model for building ontology translation systems. *Int'l Journal on Semantic Web & Information Systems* **1**(2) (2005) 22–48

---

<sup>5</sup> <http://www.neon-toolkit.org/>.

3. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-OWL: Contextualizing Ontologies. In: Proc. of ISWC 2003. Volume 2870 of LNCS., Springer (2003) 164–179
4. Haase, P., Motik, B.: A mapping system for the integration of OWL-DL ontologies. In: Proc. of IHIS 05, ACM Press (2005) 9–16
5. Chalupsky, H.: OntoMorph: A Translation System for Symbolic Knowledge. In: Proc. of KR 2000, Colorado, USA, Morgan Kaufmann (2000) 471–482
6. Dou, D., Macdermot, D., Qi, P.: Ontology translation on the semantic web. LNCS Journal of Data Semantics **2**(3360) (2004) 35–57
7. Maier, A., Schnurr, H.P., Sure, Y.: Ontology-based information integration in the automotive industry. In: Proc. of ISWC 2003. Volume 2870 of LNCS., Springer (October 2003) 897–912
8. Omelayenko, B.: RDF2T: A mapping meta-ontology for business integration. In: Proc. of Workshop on Knowledge Transformation for the Semantic for the Semantic Web (KTSW-2002) at ECAI 2002. (2002) 77–84
9. Maedche, A., Motik, B., Silva, N., Volz, R.: MAFRA - a mapping framework for distributed ontologies. In: Proc. of EKAW 2002. Volume 2473 of LNAI., Siquencia, Spain, Springer (2002) 235–250
10. Brockmans, S., Colomb, R.M., Kendall, E.F., Wallace, E., Welty, C., Xie, G.T., Haase, P.: A model driven approach for building OWL DL and OWL full ontologies. In: Proc. of ISWC 2006. Volume 4273 of LNCS., Springer (2006) 187–200
11. Silva Parreiras, F., Staab, S., Winter, A.: TwoUse: Integrating UML models and OWL ontologies. Technical Report 16/2007, Universität Koblenz-Landau (2007) Available at <http://isweb.uni-koblenz.de/Projects/twouse/tr16.2007.pdf>.
12. Parreiras, F.S., Staab, S., Schenk, S., Winter, A.: MBOTL - A Model-based Ontology Translation Language (2008) Available at <http://isweb.uni-koblenz.de/Research/MBOTL>.
13. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Satellite Events at the MoDELS 2005 Conference. Volume 3844 of LNCS., Jamaica, Springer (2005)
14. OMG: Ontology Definition Metamodel. (October 2006) Available at <http://www.omg.org/cgi-bin/doc?ptc/07-09-09.pdf>.
15. Gurevich, Y.: Sequential abstract-state machines capture sequential algorithms. ACM Trans. Comput. Logic **1**(1) (2000) 77–111
16. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL Query for OWL-DL. In: Proceedings of the OWLED 2007. Volume 258., Innsbruck, Austria, CEUR-WS.org (June 2007)
17. The Eclipse Foundation: GMT Project (2007) Available at <http://www.eclipse.org/gmt/>.
18. Jouault, F., Bézivin, J., Kurtev, I.: TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In: Proc. of 5th Int. Conf. of Generative Programming and Component Engineering, GPCE 2006, ACM (2006) 249–254
19. Atzeni, P., Cappellari, P., Bernstein, P.A.: Model-Independent Schema and Data Translation. In: Proc. of 10th International Conference on Extending Database Technology EDBT 2006. Volume 3896 of LNCS., Springer (2006) 368–385
20. Brockmans, S., Haase, P., Stuckenschmidt, H.: Formalism-Independent Specification of Ontology Mappings - A Metamodeling Approach. In: OTM 2006 Conferences. Volume 4275 of LNCS., Montpellier, France, Springer (2006) 901–908
21. Ressler, J., Dean, M., Benson, E., Dorner, E., Morris, C.: Application of ontology translation. In: Proc. of ISWC/ASWC 2007. Volume 4825 of LNCS. (2007) 830–842
22. Corcho, O., Gómez-Pérez, A.: ODEDialect: a set of declarative languages for implementing ontology translation systems. In: Int. Workshop on Semantic Intelligent Middleware for the Web and the Grid at ECAI 2004, Valencia, Spain (2004)