

Model Driven Specification of Ontology Translations

Fernando Silva Parreiras¹, Steffen Staab¹, and Andreas Winter²

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

² 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 meta model-based representation of ontology alignments that integrate semantic translations using description logics and lower level translations 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 represent and allow for translations at different layers of representation [2] [3]:

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

For semantic translations, existing frameworks provide reasoning in one or several logical paradigms, such as description logics [4] [5] or logic programming [6] [7] [8]. For lexical and syntactic translations, alignment frameworks

take advantage of platform-specific implementations, sometimes abstracted into translation patterns [9] [10] or into logical built-ins [8].

Such hybrid approaches, however, easily fail to provide clarity and accessibility to the modelers who must still get involved to achieve (near) correct translations between different ontologies. The modelers need to see and understand the semantic as well as the lexical/syntactic translations, but he can modify them only in a very intricate and disintegrated manner, drawing his attention away from the alignment task proper down into the diverging technical details of the translation model.

We propose a representation approach for ontology translation rules based on model-driven engineering (MDE) of ontology alignments. In order to reconcile semantic reasoning with idiosyncratic lexical and syntactic translations, we integrate the different layers into a representation based on a joint meta model. The joint meta model comprises description logics to specify, represent and execute semantic translation as well as OCL constraints for specifying lexical and syntactic 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. Section 5 discuss the requirements evaluation and Section 6 presents the 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). The canonical mappings covered by examples in this paper and snippets of the source and target ontologies using the Manchester OWL Syntax [11] 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 mapping. the classes `Chapter` and `InBook` in ontology #101 are translated into the class `InBook` in the ontology #303. The object property `month` having a gregorian month, e.g., ‘‘--01’’, is translated into its equivalent unabbreviated form, e.g., ‘‘January’’. The data property `pages` in ontology #303 can be calculated by subtracting the data property `initialPage` from `endPage` in ontology #101. Being *builtin: notShortened* a built-in returning the unabbreviated month, *builtin: toupper* a built-in to capitalize strings, *builtin: -* a subtractor, *s* the name space of the source ontology #101, and *t* the name space of the target ontology #303, the translation rules can be written as follows:

$$\begin{aligned} \forall x, m, t, p : t: InBook(x) \wedge t: month(x, m) \wedge t: title(x, t) \wedge t: pages(x, p) \leftarrow \\ s: Chapter(x) \wedge s: month(x, y) \wedge builtin: notShortened(y, m) \wedge \end{aligned}$$

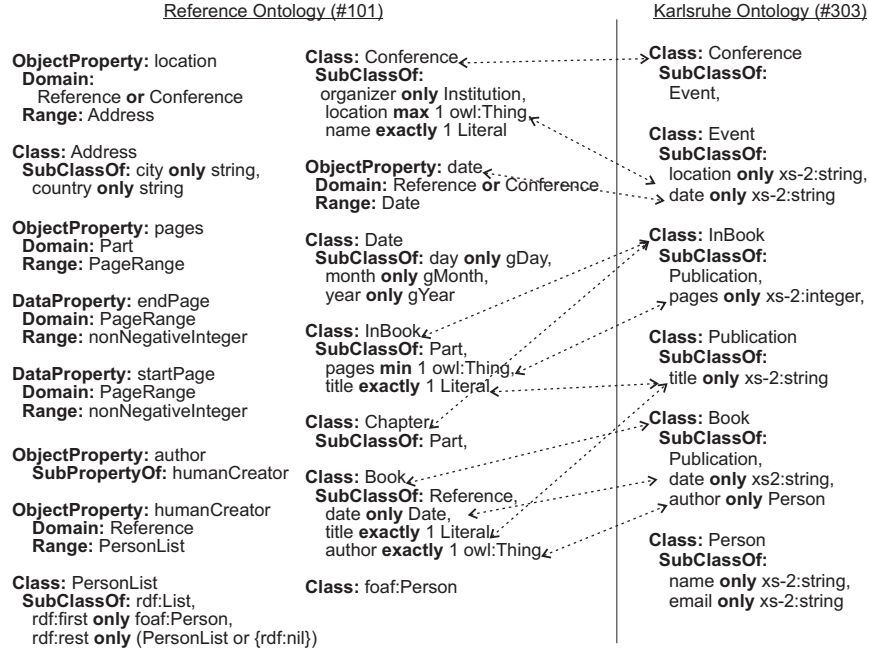


Fig. 1. Ontology mapping challenge for the running example.

$$\begin{aligned}
 & s: \text{title}(x, z) \wedge \text{builtin: toupper}(z, t) \wedge \\
 & s: \text{page}(x, w) \wedge s: \text{startPage}(w, s) \wedge s: \text{endPage}(w, e) \wedge \text{builtin: } \neg(e, s, p) \quad (1) \\
 \forall x, m, t, p : & t: \text{InBook}(x) \wedge t: \text{month}(x, m) \wedge t: \text{title}(x, t) \wedge t: \text{pages}(x, p) \leftarrow \\
 & s: \text{InBook}(x) \wedge s: \text{month}(x, y) \wedge \text{builtin: notShortened}(y, m) \wedge \\
 & s: \text{title}(x, z) \wedge \text{builtin: toupper}(z, t) \wedge \\
 & s: \text{page}(x, w) \wedge s: \text{startPage}(w, s) \wedge s: \text{endPage}(w, e) \wedge \text{builtin: } \neg(e, s, p) \quad (2)
 \end{aligned}$$

The translation rule of authors presents another problem. 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 *builtin: flatten* be the built-in able to filter a list structure into object properties, the referred rule can be written as follows:

$$\begin{aligned}
 \forall x, u : & t: \text{Book}(x) \wedge t: \text{author}(x, u) \leftarrow \\
 & s: \text{Book}(x) \wedge s: \text{author}(x, y) \wedge \text{builtin: flatten}(y, u) \quad (3)
 \end{aligned}$$

However, built-ins are black boxes that conceal knowledge about the algorithm, compromising traceability and maintenance. Therefore, an approach able to specify the rules and the built-ins without code specifics is required. We organize the requirements using three of the four layers proposed by Corcho and Gómez-Pérez [2]: the lexical layer, the syntactic layer and the semantic layer.

1. The lexical layer deals with distinguishing character arrangements, including:
 - (a) *Transformations of element identifiers*. It is required when different principles to name objects are applied, for example, when transforming the data property `title` into capital letters (cf. Rule 1).
 - (b) *Transformations of values*. It is necessary when source and target ontologies use different date formats, for example transforming a Gregorian month into an unabbreviated form (cf. 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*. It is needed when the syntax of source and target ontologies are different, e.g., when transforming from OWL 1.0 RDF/XML syntax³ into OWL 1.0 XML syntax⁴.
 - (b) *Transformations of datatypes*. It involves the conversion of primitive datatypes, e.g., converting string datatype to date datatype.
3. The semantic layer comprehends transformations dealing with the denotation of concepts over different formalisms or within the same. We consider three different aspects:
 - (a) *Inferred knowledge*. It applies reasoning services to deduce new knowledge, for example, inferring properties from class restrictions.
 - (b) *Different concepts within the same formalism*. It takes place when translating ontology elements using the same formalism, e.g, translating concepts from Karlsruhe’s OWL ontology for bibliographic references into correspondent concepts in the INRIA’s OWL ontology.

A complementary perspective of ontology translation problems is given by Dou et al. [7]. 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 remaining problems for future works.

The ontology translation approach presented in this paper relies on advances from the Model Driven Engineering (MDE) with DL semantics supporting reasoning services. With the eminence of MDE, the attention have been turned to models instead of programming code. Indeed, models are platform independent and easily extensible, i.e., to extend the expressivity of a model we just extend the metamodel. This articulation allows for specifying functions in place of built-ins, keeping off implementation specifics and allowing the modeler to concentrate on the problem domain.

3 A Model Driven Framework for Ontology Translations

This section presents the conceptual framework comprising a textual concrete syntax and an integrated metamodel as abstract syntax. We use the UML notation to specify the diagrams. Our solution relies on a layered organization of models (cf. MOF [12]).

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

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

3.1 Language Description

The following subsections present the anatomy of the translations rules following the requirements presented in Section 2. Since the translation problems are classified in non-strict layers, one rule commonly addresses more than one translation problem.

Accomplishing Translation 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 [13] to formulate queries, integrating the OCL, OWL, and MOF metamodel [14]. Indeed, OCL has successfully been used in MDE for expressing queries and side effect free operations.

Ontology translation problems at the semantic layer are treated by querying elements of the source ontology using OCL queries and matching target elements. As expected, a query is part of the input pattern in a transformation rule. Usually, a transformation rule has an input pattern and an output pattern with variables binding the elements. It is required to be able to define source and target elements and the variables. The textual syntax of the Atlas Transformation Language (ATL)[15] is appropriated for such demand.

The example depicted in Fig. 2 illustrates the textual syntax. A rule `Conference2Conference` is defined for translating the concept `Conference` in ontology #101 into `Conference` in ontology #303. The concept `Conference` is the model element to be bound to the variable `s`.

A dot-notation is used to navigate through properties. In the scope of OCL, data properties, object properties and operations are treated as properties. In this scope, a property is either a data property, an object property, a query operation or a helper.

For example, in the expression `s.location`, `s` is a reference to an instance of `Conference` with `location` resulting in a value of the type `Address`. The navigation can also end with an operation evaluation, as depict 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 support by means of employing operations or helpers. For example, the type `string` has the operation `toUpper()` returning an string object with capital letters. The evaluation of `s.title.toUpper()` capitalizes the value of the property `title`.

The set of predefined operations is available in the OCL library needed to specify constructs of the M1 layer. These operations are applicable to any type of value in OCL. For example the expression `s.owlIsInstanceOf(Misc)` evaluates to `true` if the object referred by the variable `s` has sufficient conditions to be member of the class `Misc`.

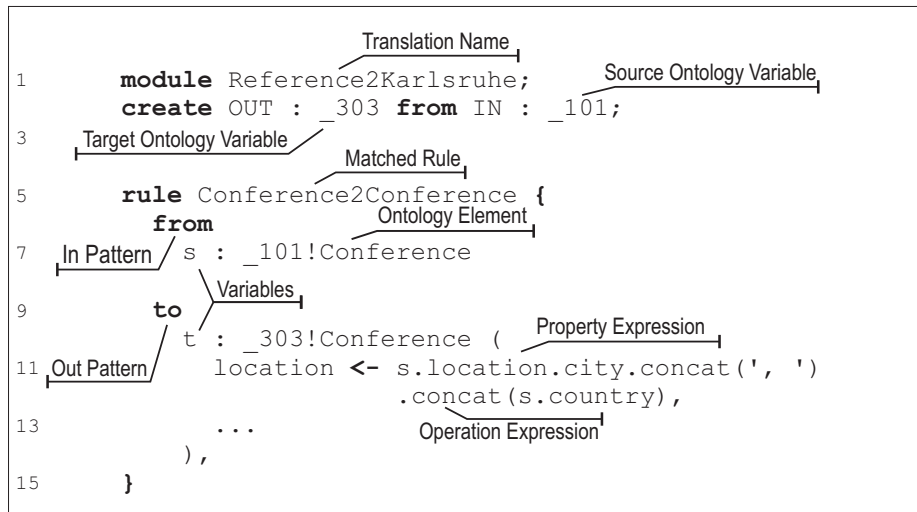


Fig. 2. Example of rule

3.2 Metamodels for ontology translations

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[12], OCL metamodel [13], OWL metamodel[16], and part of the ATL metamodel[15]. As a matter of space, we do not present the complete metamodels in this paper, but noteworthy fragments. A detailed version is available on the Internet⁵.

The translation metamodel 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 rule that has a pattern for the input model (**inPattern**) and a pattern for the output model (**outPattern**). The **InPattern** has elements that are OCL variable declarations (**Variable**). Variables are bound to model elements (**OclModelElement**). The **InPattern** has an **OclExpression** acting as filter to refine individuals of the **OclModelElement**. All these relations are depicted in Fig. 3.

Each expression in OCL has a type and the expression evaluation produces a value of type of the expression (cf. Fig. 4). The type **TUClass** is the particular composition of the OWL class with the MOF class, allowing classes to have predefined operations.

Figure 4 depicts also part of the metamodel **OclExpression** defining the abstract syntax for ODL expressions. The integration with the OWL metamodel

⁵ <http://isweb.uni-koblenz.de/Projects/twouse>

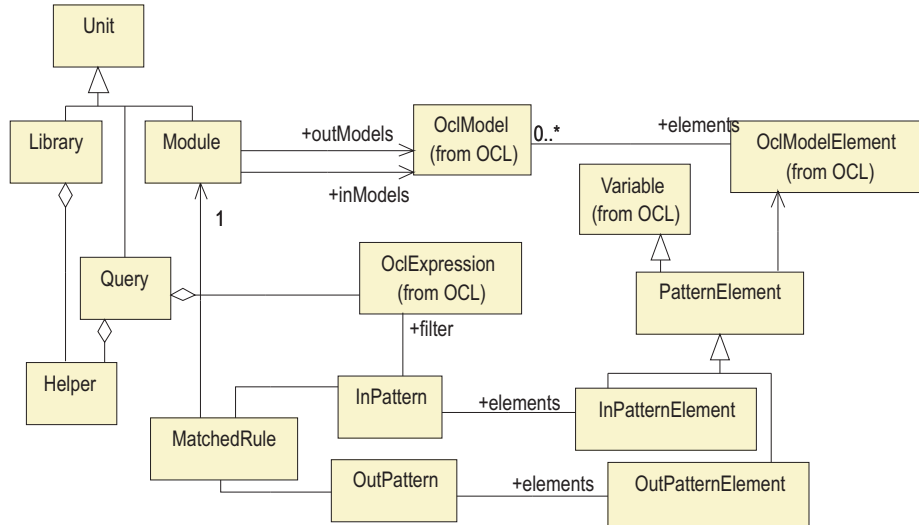


Fig. 3. Snippet of the ATL metamodel.

is accomplished by expressions of type `PropertyCallExp`. Such expression allows for navigating through OWL properties, as explained in Sect. 3.1.

Another integration point is the operation call expression (`OperationCallExp`). `OperationCallExp` 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 to invoke predefined operations or helpers. Helpers are query operations defined using OCL expressions and can be grouped into libraries.

3.3 Model Libraries

Some constructs for the ontology translation are not represented in the meta-model level (M2) but in the model level (M1) and belong to the foundation library. The foundation library is composed of the XML Schema Datatypes library, 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 metaclass. These three libraries are based on the foundation library for RDF and OWL described in the ODM specification [16].

An example of M1 object of the extended OCL library are 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 OCL library [13].

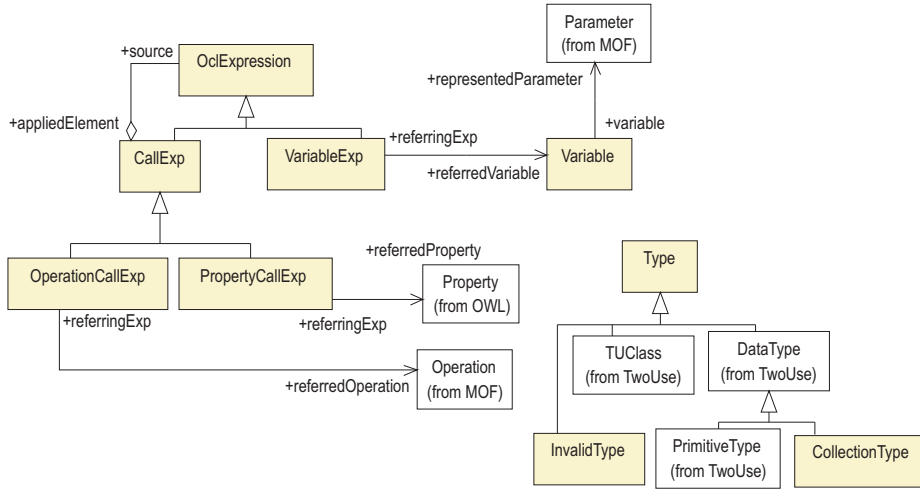


Fig. 4. Snippet of the package `Type` and package `Expressions` of the OCL metamodel

3.4 Implementation

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

We are currently implementing our solution using the Generative Modeling Technologies (GMT) project [17] under the Eclipse Modeling Framework. To specify ontology translations (1), The Textual Concrete Syntax component (TCS) [18] of the Eclipse GMT is used to specify the concrete syntax. 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 and both ontologies as source models, we use the ATL [15] transformation framework to define transformations into an ontology translations engine (2). We are currently using F-Logic [19] and Java as target transformation languages and the Ontobroker framework [20] as ontology translation solution.

Ontobroker allows for writing ontology translations using F-Logic rules and a collection of built-ins. It is possible to define new built-ins by Java programming. Elements of the ontology translation specification model concerning translation problems at the semantic layer are transformed by ATL into F-Logic rules. Part of these rules involves built-ins to handle translation problems at the lexical and syntactic layer. These built-ins 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 at Section 4, specifying the translation rules and transforming the ontology translation specification into F-Logic and Java code.

4 Application

This section presents rules integrating translations problems at semantic, syntactic and lexical layers, according the problems presented in Section 4.

Example 1: Semantic, syntactic and lexical translations. 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” (Listing 1.1). This helper is applicable only to the `gMonth` datatype.

Listing 1.1. Semantic, syntactic and lexical translations

```

1 helper context _101!gMonth
  def: notShortened() : String =
3   Sequence{'--01', '--02', '--03'} //etc.
  ->iterate(m: String; ret: Sequence =
5   Sequence{'January', 'February', 'March'} |
      if m = self.toString() then
7       ret.at(ret.indexOf(m))
      endif
9   )

11 rule ChapterInBook2Inbook {
  from
13   s : _101!Part (s.owlIsInstanceOf(Chapter) or
                  s.owlIsInstanceOf(Inbook))
15   to
      t : _303!Inbook (
17     title <- s.title.toUpper(),
      pages <- s.page.endPage - s.page.startPage,
19     month <- s.date.month.toString().notShortened(),
      )
21 }

```

In this example, the rule **ChapterInBook2Inbook** is transformed into more F-Logic rules (Listing 1.2), whereas the helper **notShortened** is transformed into Java code (Listing 1.3). Ontobroker allows for extending the predefined built-ins by writing Java code using the extension API.

Listing 1.2. F-Logic rules corresponding to **ChapterInBook2Inbook**

```

FORALL X
  X:_303#Inbook <-
  X:_101#Part and (X:_101#Chapter or X:_101#Inbook).

```

```

FORALL X, Y, Z, U
  X: _303#Inbook[_303#title->>Y] <-
  Z: _101#Part[_101#title->>U] AND
  toupper (U, Y).

FORALL X, Y, Z, P, W, U
  X: _303#Inbook[_303#pages->>Y] <-
  Z: _101#Part[_101#page->>P] AND
  P: _101#Page[_101#startPage->>W] AND
  P: _101#Page[_101#endPage->>U] AND
  - (U, W, Y).

FORALL X, Y, Z, D, M
  X: _303#Inbook[_303#month->>Y] <-
  Z: _101#Part[_101#date->>D] AND
  D: _101#Date[_101#month->>M] AND
  notShortened (toString(M), Y).

```

Listing 1.3. Java code for the helper `notShortened`

```

package com.ontoprise.builtin.example;
// imports come here.
public class NotShortenedBuiltin extends SimpleBuiltin {

  // constructor
  public NotShortenedBuiltin() {
    this.predicate = "notShortened";
    this.arity = 2;
    this.possibleSignatures = new int[][] {
      {MONTH, VARIABLE}};
  }

  // This method is called during the evaluation.
  public void receive(ITuple tuple)
    throws KAON2Exception, InterruptedException {
    String m = tuple.getArgument(0);
    String gMonth[] = new String[]{"--01", "--02", "--03"}; /...
    String month[] = new String[]{"January", "February", "March"}; /...

    for (int i = 0; i < gMonth.length; i++){
      if (gMonth[i] = m) send(month[i]);
    }
  } // receive
} // NotShortenedBuiltin

```

Example 2: Semantic and syntactic translation of complex structures. In the ontology #101, the class `Article` has the property `author` with range of type `PersonList`. `PersonList` has a property `first` with range of type `Person` and a

property `rest` with range of type `PersonList`. In the Ontology #303, the class `Article` has the property `author` as many times as many authors there are.

This rule relies on helper, able to match elements recursively. In this case, the helper algorithm must add the current value of the property `first` to the collection of authors and verify whether the value of the property `rest` is `nil`, returning in this case the collection. Otherwise, the helper is invoked until value `nil` is found.

Listing 1.4. Using an helper to transform a `rdflist` into object properties

```

helper context !_101!PersonList
2  def: flatten() : Set(!_101!Author) =
    Sequence{self}->iterate(ls; ret : Set(!_101!Author) = Sequence{} |
4      if ls = nil then
        ret
6      else
        ret->including (ls.first) and ret->including (ls.last.flatten)
8      endif
    )
10 rule Book2Book {
12   from
    s : !_101!Book (not s.owlIsInstanceOf(Proceedings))
14   to
    t : !_303!Book (
16     author <- s.author.flatten()
        ...
18   ),
}

```

Listing 1.5. F-Logic rules for `Book2Book`

```

FORALL X
  X: !_303#Book <-
  X: !_101#Book AND NOT (X: !_101#Proceedings).

FORALL X, Y, Z, A
  X: !_303#Book[_!_303#author->>Y] <-
  Z: !_101#Book[_!_101#author->>A] AND
  flatten (A, Y).

```

Listing 1.6. Java code for the helper `flatten`

```

package com.ontoprise.builtin.example;
// imports come here.
public class flattenBuiltin extends SimpleBuiltin {
//variables
String[] ret = new String[];

```

```

// constructor
public flattenBuiltin() {
    this.predicate = "flatten";
    this.arity = 2;
    this.possibleSignatures = new int [][] {
        {VECTOR, VARIABLE}};
} // flattenBuiltin

// This method is called during the evaluation.
public void receive(ITuple tuple)
    throws KAON2Exception, InterruptedException {
    RDFList l = tuple.getArgument(0);
    ret = flatten(l);
    send(ret); }

//recursive function
public String[] flatten (RDFList list){
    if (list.isEmpty){
        return ret;}
    else {
        ret[ret.length] = list.get(0);
        return flatten(list.get(1));}
} // flatten
} // flattenBuiltin

```

5 Requirements Evaluation and Discussion

In response to the requirements educed 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 17
1.(b)	converting date formats	Listing 1.1, line 19
2.(a)	converting OWL 1.0 RDF/XML to OWL 1.0 XML	Injectors
2.(b)	converting <code>gMonth</code> to <code>String</code>	Listing 1.1, line 19
3.(a)	Union of <code>Chapter</code> and <code>InBook</code>	Listing 1.1, line 13-14
3.(b)	translating <code>Book</code> to <code>Book</code>	Listing 1.4

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`

(cf. Listing 1.1) allows for converting date formats, i.e., replacing a value of `gMonth` type to the unabbreviated form.

Problems inherent in the syntactic layer are handled distinctly. While datatype conversions are achieved by invoking predefined operations, like, for example `toString()` (cf. Listing 1.1), the translation from OWL RDF/XML to OWL XML is accomplished by injectors and extractors to serialize the models.

The semantic translation of datasets between ontologies with different vocabularies but the same formalism is demonstrated by the running example. In Listing 1.4, the individuals of the class `Book` in ontology `#101`, except those individuals of the class `Proceedings` are translated into individuals of the class `Book` in ontology `#303`.

Nevertheless, our approach has some restrictions reflected by the ATL metamodel. With ATL, it is possible to realize only unidirectional translations. A bidirectional translation must be accomplished by two unidirectional translations.

6 Related Works

C-OWL [4] and the ontology mapping system proposed by Haase and Motik [5] 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 restricted.

Brockmans *et al.* [21] have proposed an MOF-based ontology mapping metamodel, mostly an abstraction of [5], and a UML profile for defining mappings at formalism independent level. However, such work does not present how ontology translation could be realized.

Actually, the metamodel for ontology mappings can be used as input to generate ontology translations. Providing that there is a MOF-based model of the mappings between two OWL ontologies conforming to a MOF-based Ontology Definition Metamodel (ODM), a MOF-based model transformation can be performed, taking as input the MOF-model of ontology mappings, the source and target ontologies, and generating a MOF-based model for the ontology translation conforming to the proposed MOF-based metamodel.

Model transformation languages like OMG Query/View/Transformation (QVT) [22] and Atlas Transformation Language (ATL) [15] allow for defining how to transform MOF-based models using declarative and imperative constructs. Although such solutions provide for constructs to address lexical and syntactic translation problems, they do not support the OWL metamodel. Our contribution extends the ATL solution by integrating with the OWL metamodel.

MAFRA [9] and RDFT [10] 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 [6] and the framework proposed by Dou [7] for ontology translation rely on First Order Language (FOL) expressiveness to specify mappings. Our approach counts on the decidable subset of FOL, the Description Logic $\mathcal{SHOIN}(\mathcal{D})$, with complete and sound automated reasoning services. Moreover, as the first solution relies on PowerLoom and the second one on Web-PDDL, translation problems of the lexical and syntactic layers are difficult to deal with.

OntoMap [8] is a mapping solution allowing for visual specification of mappings, with a limited number of translation functions. However, if new translation functions are required, they must be written at the implementation level and made available for the modeler, which is not a trivial task.

In the implementation level, Corcho and Gómez-Pérez [23] propose ODEDialect, a set of declarative languages to specify ontology translation. This is a Java-based approach, thus such approach exposes the user to the complexity of a programming language.

The work of Atzeni *et al.* [24] 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 semantic layer.

7 Conclusion

This paper presents an ongoing solution for ontology translation specification that intends to be more expressive than ontology mapping languages and less complex and granular than programming languages. The solution comprises a concrete syntax and the integration of OWL, MOF, OCL and ATL metamodels. We validate our solution against ontology translation problems grouped into three non-strict layers: lexical, syntactic and semantic.

Future Work. Future areas of investigation involves 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.

References

1. Euzenat, J.: Ontology Alignment Evaluation Initiative (Mai 2007) available at: <http://oaei.ontologymatching.org/>.
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
3. Euzenat, J.: Towards a Principled Approach to Semantic Interoperability. In: *Workshop on Ontologies and Information Sharing at IJCAI 2001*. (2001) 19–25
4. 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

5. Haase, P., Motik, B.: A mapping system for the integration of OWL-DL ontologies. In: Proc. of IHIS 05, ACM Press (2005) 9–16
6. Chalupsky, H.: OntoMorph: A Translation System for Symbolic Knowledge. In: Proc. of KR 2000, Colorado, USA, Morgan Kaufmann (2000) 471–482
7. Dou, D., Macdermot, D., Qi, P.: Ontology translation on the semantic web. LNCS Journal of Data Semantics **2**(3360) (2004) 35–57
8. 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
9. Omelayenko, B.: RDFT: 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
10. 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
11. Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wang, H.: The Manchester OWL Syntax. In: OWL: Experiences and Directions (OWLED) 2006, Athens, Georgia, USA (November 2006)
12. OMG: Meta Object Facility (MOF) Specification. (November 2005) Available at: <http://www.omg.org/cgi-bin/doc?formal/2005-11-01.pdf>.
13. OMG: Object Constraint Language Specification, version 2.0. (June 2005) Available at <http://www.omg.org/cgi-bin/doc?formal/2006-05-01.pdf>.
14. 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://www.uni-koblenz.de/~aggrimm/arbeitsberichte/arbeitsberichte_16_2007.pdf.
15. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Satellite Events at the MoDELS 2005 Conference. Volume 3844 of LNCS., Jamaica, Springer (2005)
16. OMG: Ontology Definition Metamodel. (October 2006) Available at <http://www.omg.org/cgi-bin/doc?ptc/07-09-09.pdf>.
17. The Eclipse Foundation: GMT Project (2007) Available at <http://www.eclipse.org/gmt/>.
18. Jouault, F., Bézin, 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. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. J. ACM **42**(4) (1995) 741–843
20. Fensel, D., Decker, S., Erdmann, M., Studer, R.: Ontobroker: The very high idea. In: Proc. of the 11th International Florida Artificial Intelligence Research Society Conference, AAAI Press (1998) 131–135
21. 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
22. OMG: MOF QVT Final Adopted Specification. (July 2007) Available at <http://www.omg.org/cgi-bin/apps/doc?ptc/07-07-07.pdf>.
23. 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)
24. 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