International Journal on Artificial Intelligence Tools
Vol. XX, No. X (2011) 1–32

World Scientific
www.worldscientific.com

INSTANCE MIGRATION BETWEEN ONTOLOGIES HAVING STRUCTURAL DIFFERENCES

MAXIM DAVIDOVSKY

Dept of Mathematical Modeling, Zaporozhye National University, 66 Zhukovskogo st., Zaporozhye, 69063, Ukraine
m.davidovsky@gmail.com

VADIM ERMOLAYEV

Dept of Information Technologies, Zaporozhye National University, 66 Zhukovskogo st., Zaporozhye, 69063, Ukraine
vadim@ermolayev.com

VYACHESLAV TOLOK

Dept of Mathematical Modeling, Zaporozhye National University, 66 Zhukovskogo st., Zaporozhye, 69063, Ukraine
vyacheslav-tolok@yandex.ru

Received (11 March 2011)
Revised (30 May 2011)
Accepted (Day Month Year)

Ontology instance migration is one of the complex and not fully solved problems in knowledge management. A solution is required when the ontology schema evolves in the life cycle and the assertions have to be transferred to the newer version. The problem may become more complex in distributed settings when, for example, several autonomous software entities use and exchange partial assertional knowledge in a domain that is formalized by different though semantically overlapping descriptive theories. Such an exchange is essentially the migration of the assertional part of an ontology to other ontologies belonging to or used by different entities. The paper presents our method and tool for migrating instances between the ontologies that have structurally different but semantically overlapping schemas. The approach is based on the use of the manually coded transformation rules describing the changes between the input and the output ontologies. The tool is implemented as a plug-in for the ProjectNavigator prototype software framework. The article also reports the results of our three evaluation experiments. In these experiments we evaluated the degree of complexity in the structural changes to which our approach remains valid. We also chose the ontology sets in one of the experiments to make the results comparable with the ontology alignment software. Finally we checked how well our approach scales with the increase of the quantity of the migrated ontology instances to the numbers that are characteristic to industrial ontologies. In our opinion the evaluation results are satisfactory and suggest some directions for the future work.

Keywords: ontology; structural difference; instance migration; methodology; software tool.

1. Introduction

Instance migration is an important phase in ontology engineering and management activities. A large number of ontologies describing similar domains from different

viewpoints have been developed to date. Therefore an effective re-use of their assertional knowledge is rational. Ontology instance re-use is also essential in ontology evolution. When a new ontology version is developed it is often necessary to transfer the instances of the previous version(s) to the newer version. The development of a newer ontology version starts with the implementation of the required changes in its TBox. Therefore the reuse of the ABox could be ensured if the elements of the ABox are accordingly transformed. Assertional parts of ontologies can contain a large quantity of instances that in turn makes manual transformations and instance transfer a laborous task.

The article presents our approach to semi-automated instance migration based on the use of the formal rule patterns describing transformations required for transferring instances from the source ontology to a target one. The approach is implemented in our software prototype that has been developed as a plug-in for the Project Navigator (PN) Framework for carrying out instance migration between the versions of PSI[a] ontologies[1, 2]. PSI project developed the methodology and toolset for assessing, predicting, and optimizing the performance of engineering design systems in microelectronics and integrated circuits (MIC)[3]. A multi-agent system (MAS) for holonic simulation of an engineering design system in the domain of MIC design is developed as a part of the PN Framework[4]. The MAS assists in analyzing and assessing the performance of a design system as a tool for simulation experiments.

The article also reports on the evaluation of our methodology and the prototype software tool for iterative semi-automated instance migration between the versions of industrial ontologies with evolving schemas, between distributed ontologies of moderate size with overlapping or partially matching schemas, and between ontologies of industrial size in the same domain having different schemas. The evaluation experiments were set up in line with these three use cases. The first experiment evaluates the quality of instance migration between the versions of the PSI Suite of Ontologies. PSI ontologies are used by cooperating software agents that simulate planning and execution of dynamic engineering design processes[4] in industrial MIC design domain. This use case is tailored to support the evolution of the PSI Suite of Ontologies which assertional part is used as the knowledge of a multi-agent system of cooperative software agents – in distributed settings. The aim of the evaluation experiment based on the second use case is generating reproducible migration quality measurements for the publicly available set of ontologies. For that the benchmark set of ontologies of the Ontology Alignment Evaluation Initiative – (OAEI) 2009 Campaign[b] has been used. The majority of these ontologies are artificially built using the common parent ontology by injecting different sorts of changes. By that a distributed collection of similar ontologies describing the same domain is modeled. The objective of our third evaluation experiment was to measure the performance of our tool for assessing its applicability to the industrially large volumes of instances. We have chosen two industrial ontologies with overlapping schemas that are both based on the

[a] Performance Simulation Initiative (PSI) was an R&D project (2003-2009) of Cadence Design Systems GmbH.
[b] oaei.ontologymatching.org/2009/benchmarks/bench.zip/

GoodRelations[5] ontologies as their common foundation. Both ontologies have rather simple schemas – so the transformation complexity was low. However, the number of assertions in the source ontology was big and they were available in modules. So adding these modules to the migration process incrementally allowed us to measure the execution times and to assess the computational performance of our software when working with industrial size knowledge repositories.

The article is organized as follows. Section 2 presents our motivation for undertaking this research. Section 3 provides a formal statement for ontology instance migration problem and outlines the approach to the solution. Section 4 gives an overview of the related work and outlines our contributions. Section 5 introduces our ontology instance transformation framework based on atomic transformation patterns. It is explained how instance transformation rules are designed and used for migrating ABox elements using a walkthrough example of a `Biblio` ontology. Section 6 elaborates on the implementation of our solution by presenting the algorithm and the software tool that have been developed for ontology instance migration. The architecture and the functionality of the components are explained. The use of the tool is illustrated by following the Biblio example. Section 7 presents our evaluation experiments with three different sets of ontologies. In Section 8 the results and the ways to solve problems that have been investigated in our evaluation experiments are discussed, the conclusions are drawn, and our plans for the future work are outlined.

2. Motivation

When ontologies evolve the structural changes are reflected in the ontology schemas. A new schema version is developed by implementing these changes. As a rule structural changes are carried out manually using various ontology engineering methodologies and tools[6, 7, 8]. The next step of the development of the newer ontology version is transferring the instances from the older set of ontology individuals (assertions) to the newer one. Doing this manually proves to be very laborious. The required effort appreciably increases with the number of transferable assertions. Furthermore it can be error prone.

In the PSI project we have developed the Suite of PSI Ontologies[c] that serve as domain models for Microelectronic and Integrated Circuit design. More than 20 subversions of the Suite have been developed iteratively in response to the refinements of the requirements to these knowledge representations. The assertional part of the PSI Suite incrementally received new ontology instances describing different engineering design projects. Those instances came from different sources. The minor part of them has been manually coded as test cases in the PSI and PRODUKTIV+[d] projects. Another substantial

[c] The on-line documentation may be retrieved from isrg.kit.znu.edu.ua/ontodocwiki/index.php/PSI_Suite_of_Ontologies

[d] PRODUKTIV+ (secure.edacentrum.de/produktivplus/) is the accomplished R&D project funded by the German Bundesministerium für Bildung und Forschung.

part of the assertions has been mined from the design project logs in the ACTIVE[e] project. The set of assertions describing a moderately complex project in MIC design may comprise dozens of thousands instances. Hence, performing a transition of the knowledgebase to a newer version of the ontology has become a challenging problem because of the required effort and the sensitivity to the possible transformation errors. A tool for assisting a knowledge engineer in migrating the assertions from the older ontology version to the newer one has therefore been demanded.

Very similarly, if distributed ontologies are populated based on the assertions taken from the other ontologies in the same domain[f], the patterns apply for transforming the instances according to the structural differences. However, in this case manual transformation is even more inappropriate than in the previous case of ontology evolution. It often has to happen at run time and between the software entities that own respective distributed knowledge representations. Luckily, different distributed variants of the domain ontology may be regarded as different versions of the same ontology that differ by the set of structural changes and their assertional parts. One of the experimental cases in our research deals with the migration of instances between the GoodRelations-compliant (see also www.heppnetz.de/projects/goodrelations/) industrial ontologies. This case is of particular interest because: (i) the ontologies are the collections of assertions describing real world industrial domain; and (ii) the ontologies are big enough to assess the efficiency and performance of the prototype tool.

Our objective in the research reported in this article was to develop and evaluate the prototype software tool for semi-automated instance migration from an older version to a newer version of a set of ontology modules. As explained above, the tool may also be used for merging distributed ontologies or exchanging assertions among these for mutual enrichment or harmonization.

3. Problem Statement and Solution Outline

Let us assume, as suggested by description logics[9], that an ontology O comprises its schema S and the set of individuals I.

$$O = (S, I) \tag{1}$$

Ontology schema is also referred to as a terminological component (TBox). It contains the statements describing the concepts of *O*, the properties of those concepts, and the axioms over the schema constituents. The set of individuals, also referred to as assertional component (ABox), is the set of the ground statements about the individuals and their attribution to the schema – i.e. where these individuals belong.

Let us consider two arbitrary ontologies $O_s = (S_s, I_s)$ and $O_t = (S_t, I_t)$ that conceptualize the semantics of the same universe of discourse *U*. *U* could be regarded as

[e] ACTIVE – Enabling the Knowledge Powered Enterprise (active-project.eu) is the accomplished EU FP7 Integrating Project.

[f] For example when cooperating software agents communicate their partial knowledge to their peers in a multi-agent system.

a collection of ground facts: $U = \{f\}$. Essentially, O_s and O_t are the interpretations of *U*. These ontologies would be considered identical if and only if:

$$\forall f \in U \, \text{int}_{I_s}(f) \equiv \text{int}_{I_t}(f), \tag{2}$$

where $\text{int}_I(f)$ is the interpretation of the fact *f* by the individuals from *I* of ontology *O*.

Let us introduce an abstract metric of interpretational difference $idiff(U, O_s, O_t)$. The value of *idiff* will be equal to zero for identical ontologies and will increase monotonically to one with the increase of the number of $f \in U$ such that $\neg(\text{int}_{I_s}(f) \equiv \text{int}_{I_t}(f))$. Hence, $idiff = 1$ iff $\forall f \in U \neg(\text{int}_{I_s}(f) \equiv \text{int}_{I_t}(f))$. This metric $(1 - idiff)$ is further interpreted as balanced F-measure in our evaluation experiments presented in Section 7.

Ontologies O_s and O_t are structurally different if their schemas differ: $S_s \neq S_t$. This structural difference may be presented as a transformation $\text{T}: S_s \rightarrow S_t$. If a finer grained look at an ontology schema is taken, one may consider *S* comprising the following interrelated constituents:

$$S = \{S^C, S^O, S^D, S^A\}, \tag{3}$$

where: S^C is the set of statements describing concepts; S^O is the set of statements describing object properties; S^D is the set of statements describing datatype properties; and S^A is the set of axioms specifying constraints over S^C, S^O, and S^D. One may notice that these constituents correspond to the types of the schema specification statements of an ontology representation language *L* which is used for specifying O_s and O_t. The transformation *T* may be sought in the form of nested transformation rules over the constituents of S_s resulting in the corresponding constituents of S_t.

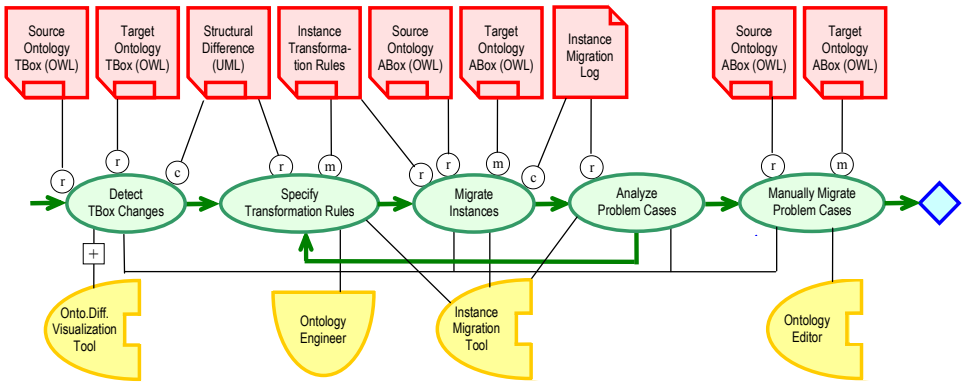
Let us assume now that, given two structurally different ontologies O_s and O_t, the ABox O_s contains individuals: $I_s \neq \varnothing$, while the ABox of O_t is empty: $I_t = \varnothing$. Therefore O_s and O_t are not identical (Eq. 2). The problem of minimizing $idiff(U, O_s, O_t)$ by: (i) taking the individuals from I_s; (ii) transforming them correspondingly to the structural difference between O_s and O_t using T; and (iii) adding them to I_t – is denoted as **ontology instance migration problem**.

Ontology instance migration problem can theoretically be solved in one-shot. In practice however, each of the sub-tasks (ii-iii) may result in the loss of assertions because of the problems discussed in Section 8. Therefore an iterative refinement of the solution could yield results with lower *idiff* value. Such a solution is presented in this article. In fact, an iterative solution of ontology instance migration problem develops a sequence of O_s states $O_s^{st_i}$ in a way to minimize the $idiff(U, O_s, O_t)$ such that:

$$idiff(U, O_s^{st_i}, O_t) < idiff(U, O_s^{st_j}, O_t) \rightarrow i > j \tag{4}$$

where: $O_s^{st_i}$ is O_s in the state after accomplishing iteration *i*; *i*, *j* are iteration numbers.

In our approach the process of ontology instance migration (Fig. 1) starts with the analysis of the changes in the TBoxes of the ontology versions. This analysis is done by a human knowledge engineer with the help of the Ontology Difference Visualizer (ODV) tool[10]. As a result the rules for transforming the instances based on the patterns for the discovered structural changes are written down by the ontology engineer. The activity of



Legend: r – read; c – create; m – modify; + – optional

Fig. 1 Instance Migration Process specified in ISO/IEC 24744 notation for describing methodologies[11].

specifying the rules is supported by the transformation rules editor that is the part of the Ontology Instance Migration (OIM) tool presented in Section 6. These rules are coded in an XML-based (www.w3.org/XML/) language and further used for guiding the automated transfer of the instances from I_s to I_t. The subsequent activity of automated ontology instance migration is performed by the OIM engine without human intervention. Instance migration results in the transfer of all the assertions that do not require the resolution of the problem cases by the ontology engineer. The cases that caused problems are recorded in the migration log. Migration log is the input information for the subsequent activity of the problem cases analysis. At this step the ontology engineer may decide that the transformation rules need to be refined in order to ensure the automated transfer in the detected problem cases in the next iteration. Otherwise the ontology engineer may decide that the remaining instances are not required in the target ABox or may be migrated manually. If so the manual transfer of the required assertions is done using an ontology editor.

4. Related Work

The predecessor work to the research presented in this article has been done by Vladimirov et al.[12]. Current research and development further extend and refine these results. As opposed to our current approach[13], the predecessor work[12] is based on the use of an ontology instance migration scenario driving the migration process and encoded as a Python script. A more comprehensive survey of alternative approaches to ontology instance migration is given in our technical report[14]. A good survey of the related work has been done by Serafini and Tamilin[15]. In addition to their overview, more frameworks for ontology evolution and ontology versioning need to be mentioned – for example the results published by Maedche et al.[16], Klein et al.[17], the Karlsruhe Ontology and the Semantic Web tool suite (kaon.semanticweb.org) as they provide some extra bits of the required functionality for ontology instance migration.

Ontology instance migration for evolving ontologies is the problem of ontology version management. As such it has not been fully researched so far. To the best of our knowledge, the literature reporting the results in this sub-field of ontology management is limited. However, looking at a broader knowledge and data change and transformation research landscape is helpful. For instance it is useful taking a look at ontology translation and database transformation research[18, 19, 20]. Ontology translation approaches can be classified as: ontology extension generation; datasets translation and querying through different ontologies. Dataset translation is of particular relevance to our work.

Making use of several heterogeneous distributed knowledge representations that describe the same or partially overlapping domains is traditionally regarded as the problem of ontology matching and alignment. This subfield of knowledge management is being extensively researched. One good review of the state of the art in this field is by Euzenat and Shvaiko[21]. Several software prototypes attempt solving ontology matching and alignment problem in a fully automated way. A popular instrument for the comparative evaluation of their results is Ontology Alignment Evaluation Initiative (OAEI). The most recent results of this benchmarking competition are dated 2010[22]. We make our results comparable to the results of the participants of this competition held in 2009[23] by using the same set of benchmark ontologies. In fact our tool does the alignment of ontological interpretations (as specified in Section 3) with better recall, precision, and F-measure than any of those tools. Indeed, the maximal recall, precision, and F-measure results of the tools that were evaluated in the web directories test case[23] of OAEI 2009 are respectively 060, 0.65, 0.63. In our evaluation experiments the results reach the level of approximately 0.83-0.9 for these metrics. The main reason is the use of human assistance in the iterations for refining and adjusting the transformation rules[g]. Though our approach requires some human effort and eventually takes longer than any of the fully automated one-shot solutions, it produces results of a substantially better quality. Because of that and despite the extra overhead of involving humans, our approach may be considered acceptable in industrial settings, for example in the management of industrial knowledge.

To conclude we have to state that fully automated ontology instance migration is the problem that remains an unsolved challenge to date. In particular, we are not informed about a tool that is capable to solve the problem reliably and with acceptable quality without human intervention. The analysis of the available literature on the tool support reveals that ontology instance migration is often carried out manually, using a tool for defining the differences between the TBoxes of the source and the target ontologies (e.g. PromptDiff[24]).

Finally, the evaluation of the quality of the results of instance migration and, hence, of the efficiency of the used methods is essential. For quality measurements in our evaluation experiments we have adapted the metrics used in data migration[25] and schema matching[26]. These metrics originate from the information retrieval[27] domain.

[g] This can not be regarded as a direct comparison to the similar solutions for ontology instance migration problem, but only as an indication of the quality of our solution. Unfortunately, at the time of writing this article we did not find the publications of the experimental results of the solutions that are similar to ours.

5. Instance Transformation Patterns and Rules

For correctly and completely migrating instances between a source and target ontology the transformation process has to be explicitly and formally specified. The specification implies an (implicit) declaration of the set of transferable individuals that is achieved by the explicit selection of the set of concepts and properties which instances have to be migrated. Also it is necessary to specify the set of required transformations over the migrating individuals. Having done that, we obtain the set of transformation rules for instance migration.

In the case of evolving ontologies the process of the creation of a new ontology version starts with applying changes to the TBox. Thus by comparing the TBoxes of respective ontology versions we can identify a certain set of structural changes conditioning differences between the versions. In the case of ontologies having overlapping domains we first have to determine the correspondences between the entities forming the schemas of these ontologies. Then, using the obtained mappings, we can similarly determine the structural changes between the ontologies.

Considering all possible kinds of changes in such a way, the set of change patterns that underlie the kinds of differences between the sets of individuals I_s and I_t can be defined – in particular such types of changes as the presence or absence of some datatype property, the occurrence of a new object property or the removal of any old object property, etc. Furthermore, based on the set of such change specifications it is possible to define the set of typical atomic transformation operation patterns over the individuals liable to migration.

Following the discussion of Eq. 3, structural transformations are defined as the mappings of ontology schema constituents:

$$\begin{aligned} T^C &: S_s^C \to S_t^C, \\ T^O &: S_s^O \to S_t^O, \\ T^D &: S_s^D \to S_t^D, \\ T^A &: S_s^A \to S_t^A \end{aligned} \tag{5}$$

specified as the sets of rules, for example: $T^C = \{t^C\}$. It needs to be mentioned that these rules may have a nested structure. Concept transformations may nest the rules from T^O, T^D, or T^A; property transformations may nest the rules from T^A. The reason for having this structure is that a transformation of a particular type of a schema constituent may also require the related transformations of its structural context represented by the related structure constituents of other types. In particular, a concept definition transformation may involve the changes in its object or datatype properties, or the constraints over its subsumption relationships. One particularly relevant example is building a transformation that specifies the change of a concept caused by the addition (or change) of an equivalence or disjointness axiom. Such a transformation in our approach is specified by nesting the corresponding t^A in the t^C.

The transformation rules for the individuals have to be derived from the structural transformations $T = \{T^C, T^O, T^D, T^A\}$ and applied to the set of assertions:

$$\begin{aligned} Tr^C &: I_s^C \to I_t^C, \\ Tr^O &: I_s^O \to I_t^O, \\ Tr^D &: I_s^D \to I_t^D, \\ Tr^A &: I_s^A \to I_t^A \end{aligned} \tag{6}$$

Tr-s as the derivatives of *T*-s will have the same nested structure.

Hence, complex transformation rules may be assembled using atomic transformation specifications that represent indivisible transformation operations. Not all atomic operations that can be thought of make sense for ontology instance migration. For example, an atomic operation of concept deletion is irrelevant because no individuals belonging to the deleted concept will be migrated. We have defined instance transformation patterns for those atomic operations that make sense. A transformation pattern is therefore denoted as a mapping $Tp : L \to L$ where L is the ontology representation language (see also Section 3). Similarly to transformation rules, transformation patterns belong to different categories that apply to different kinds of statements in L:

- $Tp^C = \{tp^C\} : L \to L$ are the patterns for concept instance transformations
- $Tp^O = \{tp^O\} : L \to L$ are the patterns for object property instance transformations
- $Tp^D = \{tp^D\} : L \to L$ are the patterns for datatype property instance transformations
- $Tp^A = \{tp^A\} : L \to L$ are the patterns for the specification of the individual constraints (axioms)

The summary of the atomic transformation patterns is given in Table 1.

Table 1. Basic instance transformation patterns supported by the OIM tool

No	Type	Pattern Comment	Prob-lems
1	Concept transformations		
1.2	addition	`<concept concept_name="name"> … </concept>`	-
		Nothing happens to instances	
1.3	renaming	`<concept concept_name="oldname">` `<rename>newname</rename> … </concept>`	-
		All the instances of the **oldname** become the instances of the **newname**	
2	Object property transformations		

2.1	deletion	`<concept concept_name="name1">`
		`<removeRelation domain="name1" range="name2">`
		`propertyName</removeRelation> … </concept>`
			-

		All instances of the `propertyName` having domain `name1` (if specified) and range `name2` (if specified) are discarded	

2.2	addition	`<concept concept_name="name1">`
		`<addRelation domain="name1" range="name2">`
		`relationName</removeRelation> … </concept>`
			+

		Nothing happens to instances	

2.3	renaming	`<concept concept_name="name1">`
		`<rename>`
		`newName</rename> … </concept>`
			-

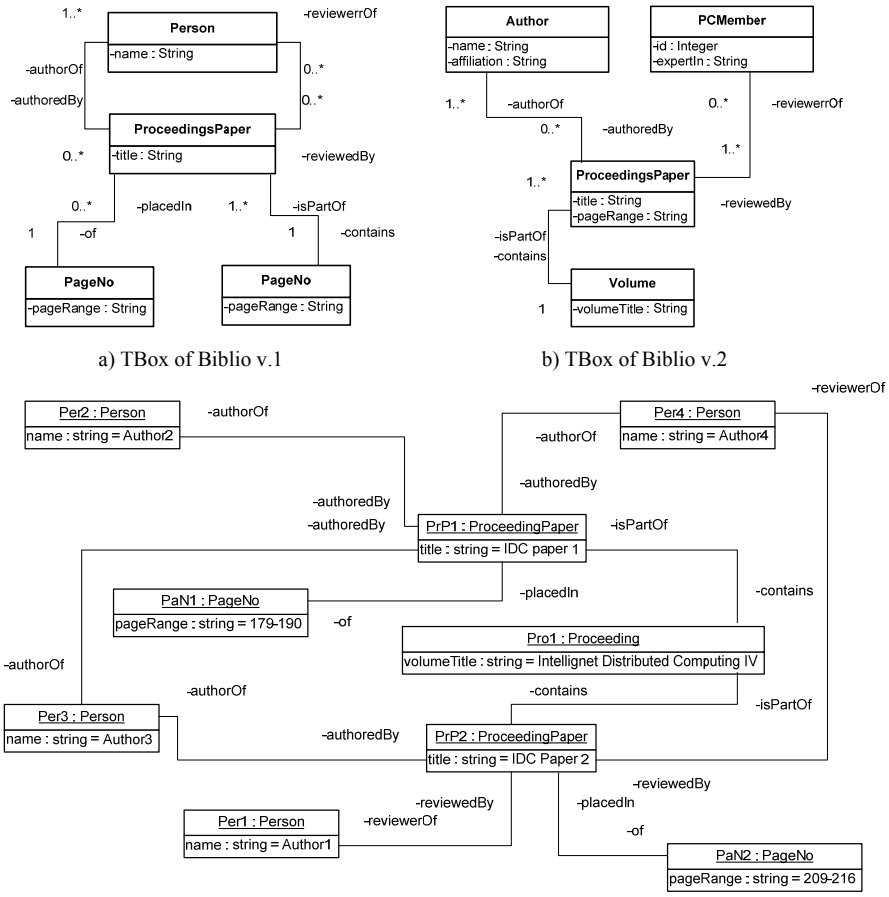
		All the instances of the `oldname` become the instances of the `newname`	
2.4	cardinality	`<concept concept_name="name1">` `<changeCardinality onProperty="propertyName">`	+

No	Type	Pattern Comment	Prob-lems
	change	**value**</changeCardinality> … </concept> Some instances of the **oldname** become the instances of the **newname**	
2.5	range change	<changeRange … ofProperty="**propertyName**" … >**name** </changeRange> Nothing happens to instances	+
3	Datatype property transformations		
3.1	deletion	<removeProperty> **propertyName** </removeProperty> All instances of the oldname are discarded	-
3.2	addition	<addProperty … datatype="**datatype value**" … value="**property value**" … toIndividual="**individual name**" …> **property name** </addProperty> Nothing happens to instances	-
3.3	renaming	<renameProperty … oldname="**name1**" … >**newname**</renameProperty> All the instances of the **oldname** become the instances of the **newname**	-
3.4	type change	<changeDatatype … ofProperty="**propertyName**" … >**datatype** </changeDatatype> All the instances of the **oldname** are transformed to the instances of the **newname** having a different data type	-

A reader may argue that Table 1 does not present the patterns for axiom transformations. Those have not been included because they do not effect in transformations, but are used for applying additional conditions on the set of individuals that is migrated. A basic pattern for specifying these conditions is the pair `<condition> …</condition>` which nests the tags for specifying conditions. For example, a condition could be a specification of a particular value of the datatype or object property (`<relation>` or `<property>` tags). Property names are specified within the tag while the value attribute allows specifying the specific value of the property or a range. A range is specified using the reserved words `%any individual%` or `%same individual%`. Constraint axioms may be specified in a similar way using the basic pattern `<constraint> … </constraint>`. Some examples are given further in the article in the discussion of the second iteration of our `Biblio` example – please refer for instance to Fig. 10.

A reader may also argue that the set of patterns in Table 1 is not complete. Our counter-argument is that the other possibly useful patterns either do not affect individuals or may be described by a nested combination of the atomic patterns listed in the table. For example, it may be noted that the operations for moving a concept to another package (ontology module) have to be also introduced. However, referencing of some concept to another package does not entail the changes in individuals. Similarly, the fact of referencing some concept as a subclass of another class also does not directly imply the changes in individuals. Probable incurred changes (e.g. inheritance of additional properties) can be described by a combination of the atomic transformation patterns.

Instance transformation rule patterns are of two kinds. The first kind is for the declaration of the set of classes which instances are liable to migration. The second represents typical transformation operations.



a) TBox of Biblio v.1

b) TBox of Biblio v.2

c) ABox of Biblio v.1

Fig. 2 The fragments of a `Biblio` ontology in versions 1 and 2.

Transformation patterns are defined using XMLSchema (www.w3.org/XML/Schema). The use of the patterns enables decoupling from particular ontology contexts and can be instantiated in a set of the specific transformation operations for the particular pair of ontologies. In our approach transformation rules are serialized in XML. This design choice allows building convenient and yet computationally efficient formal descriptions for the transformations.

The transformation rules are specified for the concrete pair of ontologies by applying the described patterns as the templates. It is done by combining appropriate transformation patterns and filling in the slots with concrete values. Those rules are further applied for performing required migration operations on the instances of the source ontology. As a result the target ontology ABox is obtained. For illustrating this approach we use a very simple and artificial example of a fragment of a `Biblio` ontology (Fig. 2).

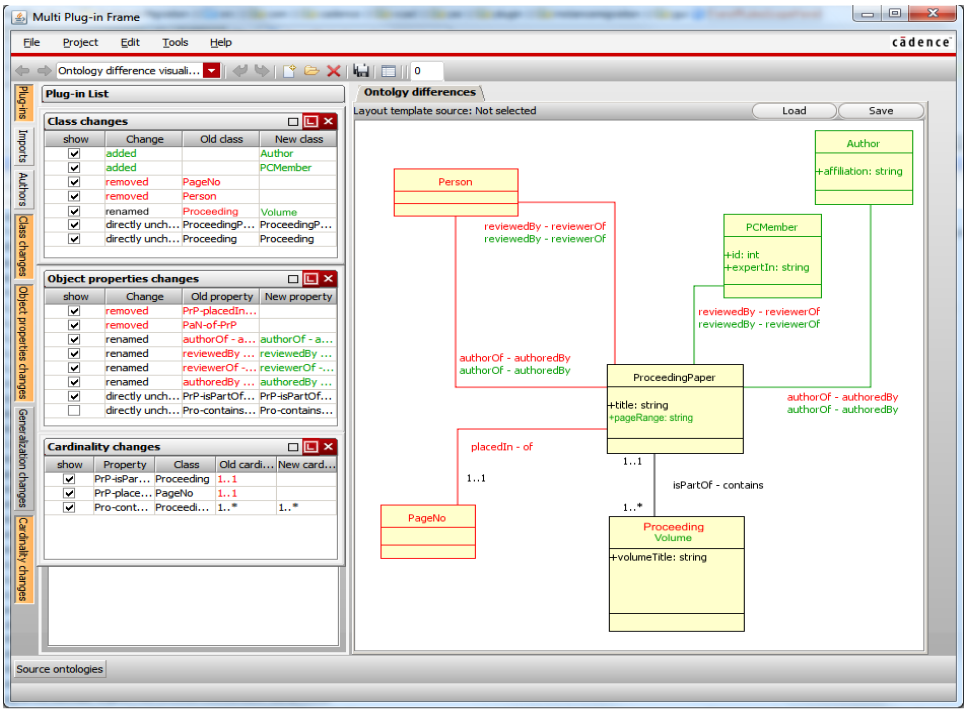


Fig. 3 The structural difference diagram of the versions of the `Biblio` ontology (Fig. 2) generated by the ODV tool.

The structural difference between `Biblio_v.1.owl` and `Biblio_v.2.owl` as discovered using the ODV tool (Fig. 3) is interpreted as summarized in Table 2. The transformation rules for those structural change elements have been coded based on the templates provided by the transformation patterns (Table 1) as shown in Fig. 4.

At the step of formulating the transformation rules for a particular migration case the names of the particular concepts and properties, and the set of transformation operations

```
<concepts
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="TRules-Biblio-v11-v22.xsd">
 <concept concept_name="ProceedingPaper">
   <addProperty data_type="string" value=""
    toIndividual="">pageRange</addProperty>
   <renameRelation domain="ProceedingPaper" range="Proceeding"
    oldName="PrP-isPartOf-Pro">PrP-isPartOf-Vol</renameRelation>
   <changeRange ofProperty="PrP-isPartOf-Vol">Volume</changeRange>
   <removeRelation domain="ProceedingPaper" range="PageNo">
    PrP-placedIn-PaN</removeRelation>
 </concept>
 <concept concept_name="Proceeding">
   <rename>Volume</rename>
   <renameRelation domain="Volume" range="ProceedingPaper"
    oldName="Pro-contains-PrP">Vol-contains-PrP</renameRelation>
 </concept>
</concepts>
```

Fig. 4 Instance transformation rules for the first iteration of the `Biblio` example.

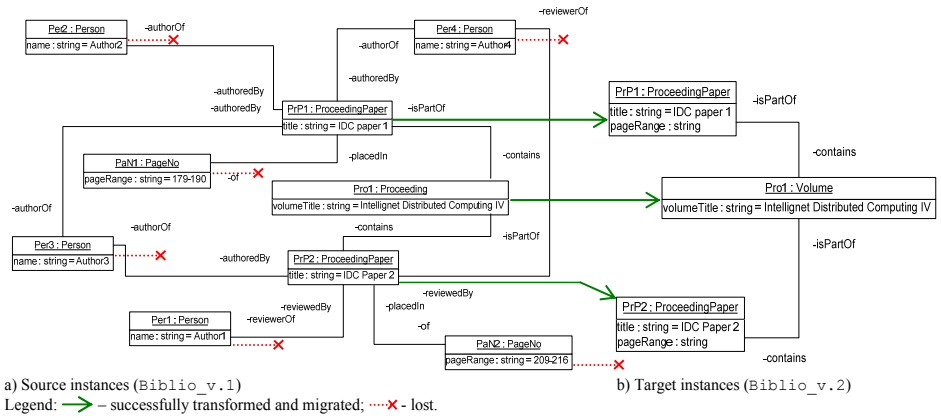


Fig. 5 The results of instance migration for the `Biblio` example in iteration 1.

that should be executed over their individuals are specified in the corresponding XML file. A root statement `<concepts>` in the XML Schema file is defined for that. This root element contains the descriptions of the concepts and operations for the particular pair of ontologies. It is formalized as a set of `<concept>` statements with respective properties that contain as their values the names of classes which instances are liable to migration. The tag `<concept>` contains the embedded tags that correspond to operations which may also have attributes. For example the `addProperty` tag has the `data_type` attribute for the added property and `value` attribute that contains the value of the property. The rules can be specified for transforming the properties of particular individuals using the `toIndividual` attribute specifying the name of the individual.

For our `Biblio` example the application of the transformation rules specified in Fig. 4 results in the ABox migration as pictured in Fig. 5. A brief analysis of these results reveals that the majority of instances of `Biblio_v.1` have been lost. Hence, straightforwardly following the detected differences between the TBoxes does not ensure the maximal possible completeness of the transfers at the ABox level. Indeed, all the author-related information has been lost because we discarded the concept of a `Person` instead of more accurately taking care of its instances. The instances of a `Person` could in fact have been separable by the difference in their relationship to the concept of a `ProceedingPaper`. So, they could have been migrated to the appropriate newly introduced concepts of an `Author` or a `PCMember`. More generally speaking, combining discarding former concepts or properties with introducing new ones in transformation rule sets most evidently leads to the losses in instances. As it will be shown further in the article, the best way for preserving the required instances is using change patterns like `<rename ...>` or `<changeRange ...>`. The rules based on `<cardinalityChange ...>` may also cause losses in migration that may be resolved using heuristics or additional rules for individual instances – as explained further in more detail.

Table 2. Summary of changes related to the transformation patterns

Transformation Pattern	Structural Change
`---`	Concepts `Person` and `PageNo` are discarded[h] together with all their object properties
`---`	Concepts `Author` and `PCMember` are introduced[i]
`<rename …>`	Concept `Proceeding` is renamed to `Volume`
`<renameRelation …>, <changeRange …>`	Object property `isPartOf – contains` is modified by relating the renamed concept (`Volume`) to a `ProceedingPaper`
`---`	The cardinality of the object property `ProceedingPaper – authoredBy–Author` is changed to `1..M`
`<removeRelation …>`	Object property `placedIn` – of a `ProceedingPaper` is discarded
`<addProperty …>`	Datatype property `pageRange` is introduced for a `ProceedingPaper`

The objectives we had when developing the methodology for instance migration and the supporting OIM tool were: (i) to provide an ontology engineer with maximal possible advise on instance losses; and (ii) to support his productive work in the iterations of the transformation rule refinement process. The details of the implementation of the OIM tool are presented in the next Section.

6. The Algorithm and Tool for Ontology Instance Migration

The algorithm for solving the ontology instance migration problem, as denoted in Section 3, has been developed (further referred to as the OIM algorithm). The component-level outline of the OIM algorithm is presented in Fig. 6. Essentially the OIM algorithm determines the functionality of the Instance Migration Engine presented further in this Section as a part of the OIM plug-in architecture (see also Fig. 8). The software components involved in the execution of the instance migration from O_s to O_t are pictured in more detail in Fig. 7. These components are also rendered bold in Fig. 6. The specification of the OIM algorithm uses the following symbols:

- $O_s = \{o_s\}$ – the set of source ontology modules (possibly coupled using import statements)
- $O_t = \{o_t\}$ – the set of target ontology modules (possibly coupled using import statements)
- $S_i^C = \{s^C\}$ – the set of statements describing concepts (classes) of a source ontology module o_i (where $int\ i = \overline{1, n_{O_s}}$, n_{O_s} – the number of source ontology modules in O_s)
- $S_i^O = \{s^O\}$, $S_i^D = \{s^D\}$ – the set of statements describing object and datatype properties in o_i respectively

[h] We do not offer a transformation pattern for discarding classes or properties as nothing has to be done with the instances of these structural elements. These instances are not taken into account in the migration process.
[i] The rules for adding concepts and cardinality change are not introduced (c.f. Fig. 4) – we have no instances for these transformed structural elements.

```
For ( r : R ) do
   //* Process the next transformation rule
  {TRProcessor: Get r from R ;
   TRProcessor: Get the value of the concept_name slot of r ;
   //* find a class with the concept_name
   For ( o_i : O ) do
      //* Process i-th ontology module
     {For ( s^C : S_i^C ) do
         //* Process the next concept and related properties
        {OntologyManager: Get s^C from S_i^C where s^C.name = concept_name ;
         DataFactory: Get I_sC according to the chosen ABScope;
         OntologyManager: Infer the sets of axioms (S̃_s^A)_i and (S̃_t^A)_k that either
         constrain or expand I_sC :
         (o_s)_i → (o_t)_k and
         (S̃_s^A)_i ⊆ (S_s^A)_i, (S̃_t^A)_k ⊆ (S_t^A)_k : ∃T_i : (S_s^C)_i → (S_t^C)_k ∨ (S_s^O)_i → (S_t^O)_k ∨ (S_s^D)_i → (S_t^D)_k ,
         int k = 1,n_Ot , n_Ot - number of target ontologies;
         Core: Query Reasoner to infer the constrained or expanded set
         of individuals Ĩ_sC based on (S̃_s^A)_i and (S̃_t^A)_k ;
         LogFactory: Write the results of the query to the Log for information;
         DataFactory: Retrieve Ĩ_sC ;
         For ( op : Op_rj )
            //* Execute the next atomic transformation operation
           {Get op ;
            Core: Generate the axioms for applying the structural
            transformations incurred by op ;
            Core: Query Reasoner to check the perspective status of the
            ontology (integrity; OWL-DL; OWL-Full) after the transformations
            are applied;
            LogFactory: Write the results to the Log;
            Core: Query Reasoner for possible additional structural
            transformations according to (S̃_s^A)_i и (S̃_t^A)_k ;
            Core: Query Reasoner to check the perspective status of the
            ontology;
            LogFactory: Write the results to the Log;
            OntologyManager: apply structural transformations;
            For ( ind : Ĩ_sC )
               //* Execute the atomic transformation operation over
                   the next individual
              {Retrieve ind ;
               Core: Generate TransformationAxioms that implement op over ind ;
               OntologyManager: Apply TransformationAxioms;
              }
            }
            LogFactory: Write the results to the Log;
           }
        }
        LogFactory: Write the results to the Log;
       }
     }
```

Fig. 6 The component-level outline of the OIM algorithm.

- $S_i^A = \{s^A\}$ – the set of constraint axioms in o_i
- $I_i = \{ind\}$ – the set of individuals of ontology o_i
- *ABScope* – the parameter that circumscribes the part of the source ABox in terms of ontology modules involved in the migration process – may be set manually
- $T = (T^C, T^O, T^D, T^A) = \{t\} = \{\{t^C\}, \{t^O\}, \{t^D\}, \{t^A\}\}$ is the set of structural transformation rules specified for a particular migration case
- $Op_{r_j} = \{op\}$ – the set of atomic transformation operations of which the corresponding transformation rule is composed (where *int* $j = \overline{1, n_T}$, n_T – the number of transformation rules). Each operation is an instantiation of a respective transformation pattern (Tp)

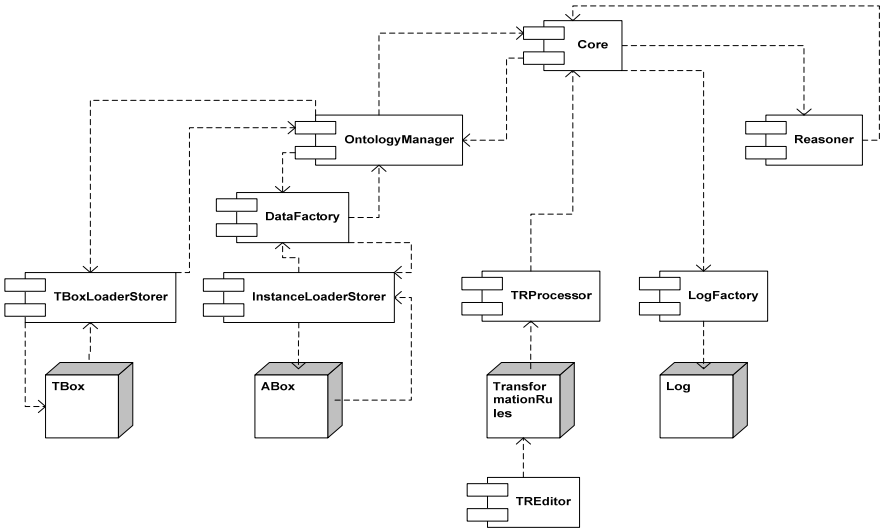


Fig. 7 The components of the Instance Migration Engine. The dashed arrow lines represent the flows of information.

The computational complexity of the OIM algorithm (Fig. 6) could be estimated as linear ($O(\|I_s\|)$) in the assumption that the number of TBox elements is substantially smaller than the number of ontology assertions. This assumption is valid for many ontologies. In particular, it is true for all the ontologies that have been used in our evaluation experiments (Section 7). This estimation is also confirmed by the results of our third evaluation experiment (Table 10).

The OIM tool is implemented as a Java (www.oracle.com/technetwork/java/) plug-in for the PN framework. The architecture of the OIM plug-in is sketched in Fig. 8. It has been developed using the IntelliJIDEA Integrated Development Environment v.8.1 (www.jetbrains.com/idea/). Java Application Programming Interface (API) for XML Processing (JAXP, jaxp.java.net) has been used for creating and processing virtual XML documents, in particular parsing tags and attributes in the transformation patterns and

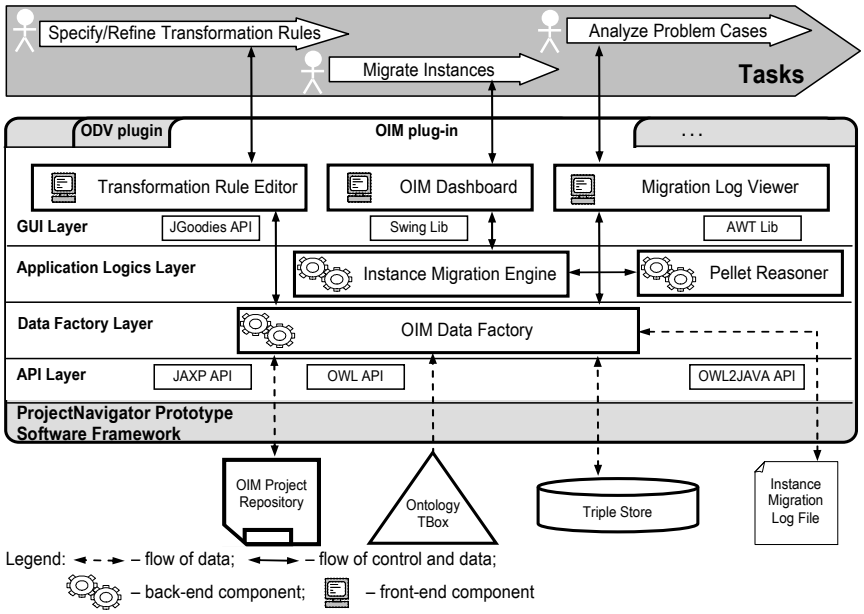


Fig. 8 The architecture of the OIM plug-in. The component structure of the Instance Migration Engine is detailed in Fig. 7.

rules. The OWL API v.2.2.0 (owlapi.sourceforge.net) has been used for processing ontology files (modules) regarding ontology modules as consistent sets of axioms and assertions. At the higher level of the representation of ontology elements the OWL2JAVA API (developed as a part of the PN framework) has been used for making the processing of the concept and instance representations better structured and more convenient in coding. Using OWL2JAVA allowed us natively assessing ontology elements as Java classes, attributes, and methods instead of working with lower-level statements or triples. Consequently, the use of this intermediate level data access factory facilitated to better decoupling from particular OWL serializations.

The Graphical User Interface (GUI) of the OIM tool has been developed according to the look and feel of the PN framework. The forms and the desktop layouts have been developed using JGoodies API (www.jgoodies.com). In addition the Swing (download.oracle.com/javase/1.5.0/docs/guide/swing/) and AWT (download.oracle.com/javase/1.5.0/docs/guide/awt/) Java libraries have been used. The Dashboard and the Transformation Rule Editor user interface of the OIM tool are pictured in Fig. 8.

We now illustrate how the OIM tool can be efficiently used for ensuring quality and completeness of instance transformations by describing the use of the tool in the second iteration of the instance migration process for our `Biblio` example. The first step an ontology engineer has to undertake after doing automated migration of ontology instances is the analysis of the problems encountered in this migration (c.f. the methodology, Fig. 1). This activity is supported by the OIM Migration Log Viewer

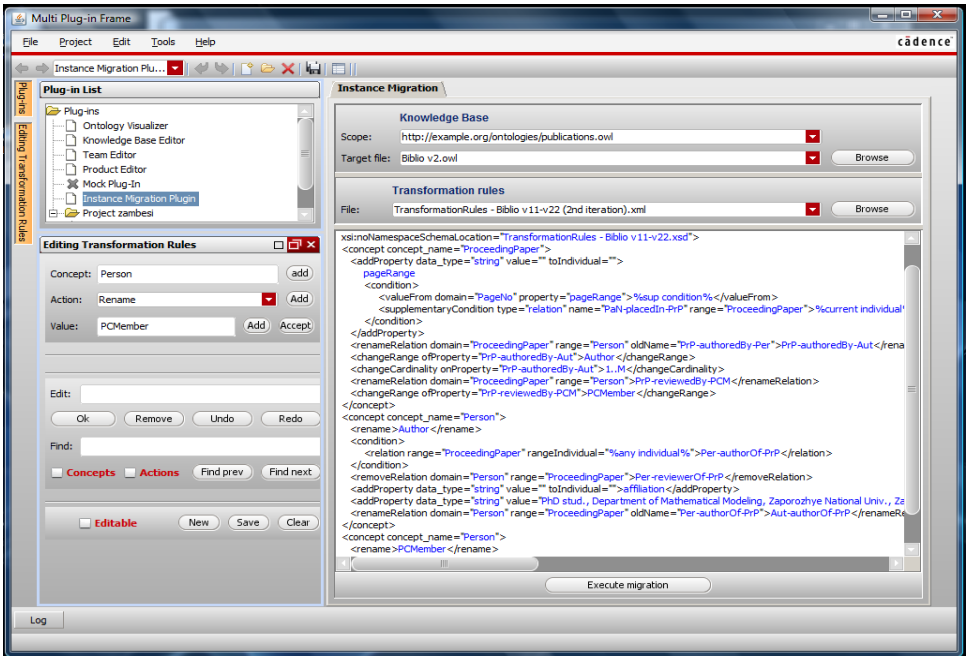


Fig. 9 OIM tool Dashboard and Transformation Rule Editor (`Biblio` example, iteration 2).

functionality. The viewer displays the records about the problems encountered at the migration execution time in the order of their appearance. This order is also a useful piece of information because it allows assessing how the combination of the basic operations in the given sequence led to the migration problems. The problems encountered at iteration 1 for our `Biblio` example are listed in Table 3 in the same order the corresponding records had appeared in the migration log.

Table 3. The analysis of ABox changes after the first iteration based on the Migration Log

Individuals		**Migration Problems**	**Lost Information**
Ver.	**Name**		
v.1 v.2	`PrP1:ProceedingPaper` `PrP1:ProceedingPaper`	added an empty slot of property `<pageRange>`	value of `<pageRange>`
		object property `<PrP-authoredBy-Per>` is deleted	object property `<PrP-authoredBy-Aut>`
		object property `<PrP-reviewedBy-Per>` is deleted	object property `<PrP-reviewedBy-PCM>`
v.1 v.2	`PrP2:ProceedingPaper` `PrP2:ProceedingPaper`	Same to `PrP1: ProceedingPaper`	
v.1 v.2	`PaN1:PageNo` ---	Datatype property values lost when the instances of PageNo were discarded	value of `<pageRange>` property: "179-190"
v.1 v.2	`PaN2:PageNo` ---	Same to `PaN1: PageNo`	value of `<pageRange>` property: "209-216"

Individuals		Migration Problems	Lost Information
Ver.	**Name**		
v.1 v.2	Per1:Person ---	The instance of a Person that was related to the instance of a ProceedingPaper as the reviewer was erroneously discarded	Per1:PCMember
v.1 v.2	Per2:Person ---	The instance of a Person that was related to the instance of a ProceedingPaper as the author was erroneously discarded	Per2:Author
v.1 v.2	Per3:Person ---	The instance of a Person that was related to the instance of a ProceedingPaper as the author was erroneously discarded	Per3:Author
v.1 v.2	Per4:Person ---	Same to Per2: Person	Per4:Author
v.1 v.2	Per4:Person ---	Same to Per1: Person	Per4:PCMember

The result of the analysis of the encountered problems supports our suggestions about the use of the transformation patterns given at the end of Section 5. Indeed the losses of the concept and property instances were caused by a straightforward way of following the discovered structural difference between the TBoxes of Biblio_v.1 and Biblio_v.2 provided by the ODV tool (Fig. 3). Following these suggestions we discarded the concepts and the properties attached to these concepts with relevant instances. We then added new concepts and properties. Hence, the instances listed in the last right column of Table 3 have been lost.

The task of a knowledge engineer at this step is to decide if the lost instances are still required in the target ABox and, if so, to refine the transformation rules in order to migrate these instances at the next run of the Instance Migration Engine. In the Biblio example the knowledge engineer decided to do the refinements pictured in Fig. 10. These changes have been done in line with our recommendations about combining basic transformation patterns. Instead of discarding the instances of a Person the refined rules make a separation of these instances by their object properties. Those which care the … authorOf … property are migrated to the newly introduced concept of an Author. Those related to an instance of a ProceedingPaper as … reviewerOf … become the instances of a PCMember.

In addition, the example set of the refined rules demonstrates how default property values may be assigned to the new properties in the target ABox. For example, the value "PhD stud., Dept1, Uni1" is assigned to the newly introduced affiliation property of the Per_3 instance of an Author. Default properties may also be assigned to a group of instances. For that the conditions for determining the needed set of instances by using the <condition …> pattern have to be specified. In the case of adding of a default value to all the instances of some concept it is specified in the value slot. The conditions and the identification of a particular instance are left blank.

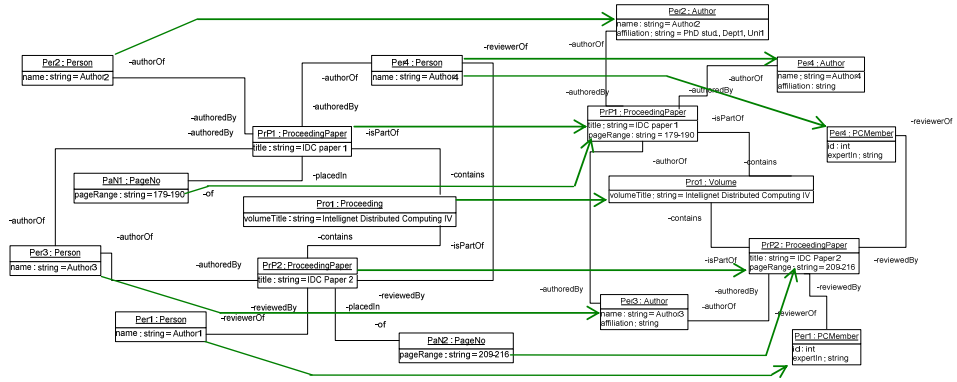
OIM Transformation Rule Editor (Fig. 9) is used to support the specification or refinement of the transformation rules. It provides a template based interface that supports all the transformation patterns listed in Table 1. The rules are specified by filling

```
<concepts
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="TransformationRules-Biblio-v11-v22.xsd">
 <concept concept_name="ProceedingPaper">
  <addProperty data_type="string" value="" toIndividual="">pageRange
   <condition>
   <valueFrom domain="PageNo" property="pageRange">%sup condition%</valueFrom>
   <supplementaryCondition type="relation" name="PaN-placedIn-PrP"
    range="ProceedingPaper">%current individual%</supplementaryCondition>
   </condition>
  </addProperty>
  <renameRelation domain="ProceedingPaper" range="Person"
   oldName="PrP-authoredBy-Per">PrP-authoredBy-Aut</renameRelation>
  <changeRange ofProperty="PrP-authoredBy-Aut">Author</changeRange>
  <changeCardinality onProperty="PrP-authoredBy-Aut">1..M</changeCardinality>
  <renameRelation domain="ProceedingPaper" range="Person">PrP-reviewedBy-PCM
  </renameRelation>
  <changeRange ofProperty="PrP-reviewedBy-PCM">PCMember</changeRange>
 </concept>
 <concept concept_name="Person">
  <rename>Author</rename>
  <condition>
  <relation range="ProceedingPaper" rangeIndividual="%any individual%">
   Per-authorOf-PrP</relation>
  </condition>
  <removeRelation domain="Person" range="ProceedingPaper">Per-reviewerOf-PrP
  </removeRelation>
  <addProperty data_type="string" value="" toIndividual="">affiliation
  </addProperty>
  <addProperty data_type="string" value="PhD stud., Dept1, Uni1"
   toIndividual="Per_3">affiliation</addProperty>
  <renameRelation domain="Person" range="ProceedingPaper"
   oldName="Per-authorOf-PrP">Aut-authorOf-PrP</renameRelation>
 </concept>
 <concept concept_name="Person">
  <rename>PCMember</rename>
  <condition>
   <relation range="ProceedingPaper" rangeIndividual="%any individual%">
    Per-reviewerOf-PrP</relation>
  </condition>
  <removeRelation domain="Person" range="ProceedingPaper">Per-authorOf-PrP
  </removeRelation>
  <removeProperty>name</removeProperty>
  <addProperty data_type="int" value="">id</addProperty>
  <addProperty data_type="string" value="">expertIn</addProperty>
  <renameRelation domain="Person" range="ProceedingPaper"
   oldName="Per-reviewerOf-PrP">PCM-reviewerOf-PrP</renameRelation>
 </concept>
</concepts>
```

Fig. 10 Instance transformation rules for the second iteration of the `Biblio` example.

the slots of the transformation patterns with the required specific names or values of the concepts or properties. It came out of our experience in the evaluation trials that the template approach facilitates to substantially decreasing the effort spent in this step and lowering the chances of making mistakes.

The results of the subsequent Instance Migration Engine execution in the second iteration for the `Biblio` example are pictured in Fig 11. The figure shows that all the instances of `Biblio_v.1` have been correctly transformed and migrated `to Biblio_v.2` using different combinations of the transformation patterns in a proper sequence. Besides that, the default instances for the newly introduced properties have been assigned.



a) Source instances (`Biblio_v.1`) b) Target instances (`Biblio_v.2`)

Legend: → – successfully transformed and migrated.

Fig. 11 The results of instance migration for the `Biblio` example in iteration 2.

7. Evaluation

The methodology for ontology instance migration and the implemented OIM tool prototype have been evaluated with respect to the quality and completeness of the executed instance migration. We also evaluated the computational performance of the Instance Migration Engine in order to get an indication about the applicability of our approach to industrially realistic quantities of assertions. Schematically the experimental set-up is represented in Fig. 12.

Three evaluation experiments with different ontologies have been performed. In our first experiment the excerpt of the PSI Suite of Ontologies v.2.0 and v.2.2 has been used. The objective of this experiment was to reveal possible errors at migration run time and prove the concept. The objective of the second experiment was to obtain the quality measures that can be compared to the results of the automated ontology alignment software. Therefore a broader set of publicly available test case ontologies has been used.

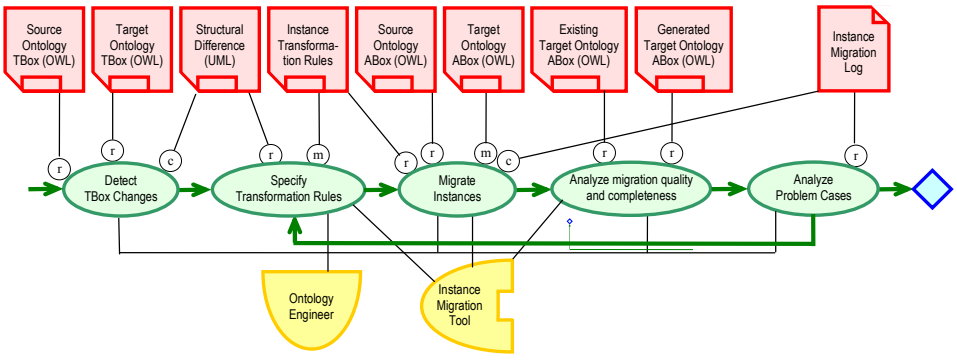


Fig. 12 The set-up of the evaluation experiments

The bigger quantity of used ontologies allowed receiving statistically more precise evaluation measurements using the metrics described below. In the experiments 1 and 2 the results of the migration have been compared to the available target ABoxes. The quality and completeness of instance migration were measured. In the third trial we have experimented with the ontologies that have been developed and are in use in real industrial settings.

For measuring the quality and the completeness of ontology instance migration *Precision* and *Recall* metrics have been adopted from the Information Retrieval domain[27] and further adapted to fit the context of our problem. In our case *Precision (P)* is the fraction of migrated individuals that are relevant. *Recall(R)* is the fraction of relevant individuals that are migrated. The contingencies are explained in Table 4 where *P* and *R* are computed as: $P = tp/(tp + fp)$; $R = tp/(tp + fn)$. The effectiveness of the migration tool can also be measured using the *Accuracy* metric. In our case *Accuracy (A)* is defined as $A = (tp + tn)/(tp + fp + fn + tn)$. An ideal migration outcome is when *Precision*=*Recall*=1. However, neither *Precision* nor *Recall* separately provides a complete picture of the correctness of the obtained results. For that the *F-measure* could be of interest as it brings *Precision* into correlation with *Recall* as a weighted harmonic mean of the both: $F = 1/(\alpha(1/R) + (1-\alpha)(1/R)) = ((\beta^2 + 1)PR/(\beta^2 P + R))$, where $\beta^2 = (1-\alpha)/\alpha$, $\alpha \in [0,1]$. If both precision and recall are of equal importance they can be equally weighted in *F* by having $\alpha = 1/2$ or $\beta = 1$. This case is the one of a *balanced F-measure* $F_{\beta=1} = 2PR/(P + R)$.

Table 4. Instance migration contingency table

	Relevant	**Nonrelevant**
Migrated	true positives (*tp*)	false positives (*fp*)
Not migrated	false negatives (*fn*)	true negatives (*tn*)

In the first evaluation experiment with the excerpt of the PSI Suite of Ontologies the total number of instances was 1890 structured in 12 ontology modules. The instances have been migrated in one shot. The contingencies for the first experiment are given in Table 5 and the results are summarized in Table 6.

Table 5. The contingency table for the excerpt of the PSI Suite of Ontologies

	Relevant	**Nonrelevant**
Migrated	360	2
Not migrated	48	1480

Table 6. The summary of the results for the excerpt of the PSI Suite of Ontologies

	Measures			
Iteration	**Precision**	**Recall**	**Accuracy**	**Balanced F-measure**
1	0.994475138	0.881632653	0.973373303	0.93466032

The source for the second experiment was the set of the test ontologies of the OAEI 2009. The choice was motivated by the public accessibility of the test set that allows cross-evaluation. 40 ontologies have been chosen which in our opinion were the most appropriable for the instance migration evaluation. The total number of instances was 2060. The instance migration process has been done in two iterations with a slight refinement of the transformation rules in the second iteration. The contingencies for the second experiment are given in Table 7 and the results are summarized in Table 8.

Table 7. The contingency table for the OAEI 2009 set of ontologies

	Relevant	**Nonrelevant**
Migrated	1475	7
Not migrated	309	269

Table 8. The summary of the results for the OAEI 2009 set of ontologies

	Measures			
Iteration	**Precision**	**Recall**	**Accuracy**	**Balanced F-measure**
1	0.99527665	0.82679372	0.84660194	0.90324556
2	0.99732381	0.98415493	0.98162729	0.99069561

The results given in Table 8 indicate that opting to an iterative approach in our methodology was correct in terms of further increasing the quality of instance migration. Indeed, the refinement of the transformation rules in the second iteration allowed increasing the recall and accuracy of our results substantially. Consequently the balanced F-measure has been improved by almost 10 percent.

In the third experiment the dataset of the European building and construction materials market for the Semantic Web (BauDataWeb, semantic.eurobau.com) has been selected as a source ABox. It is based on the freeClass Ontology for construction and building materials and services (www.freeclass.eu) that has been used as a source ontology TBox. This ontology is defined using the GoodRelations Web Vocabulary for E-Commerce (www.heppnetz.de/projects/goodrelations/) as its foundational level. The web ontology for products and services eClassOWL[5] (see also www.heppnetz.de/projects/eclassowl/) which is also based on the GoodRelations has been chosen as the target ontology. By choosing this pair of ontologies we model the semantic data integration scenario in industries. Indeed, the BauDataWeb is a very big and detailed set of assertions describing products for the construction industry. The eClassOWL ontology is also the ontology of products but with a much broader scope. The shortcoming of the eClassOWL is that it contains much less information about the products for the construction industry than the BauDataWeb ontology. Having the instances of BauDataWeb migrated to the eClassOWL enriches the eClassOWL ontology. The BauDataWeb is structured in several increments that can be processed separately. In total 100 increments of this ontology are available which contain over 60 million instances.

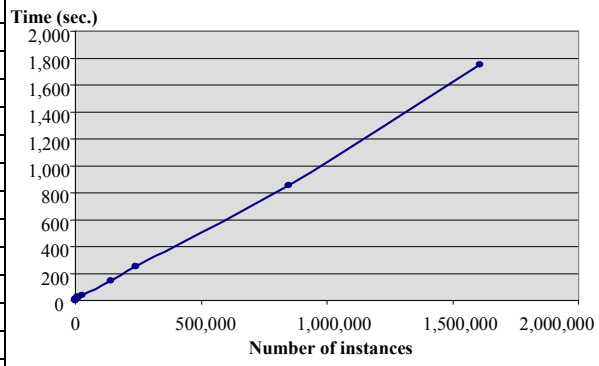
As the structural differences between the freeClass and the eClassOWL ontology schemas are very simple and cause no problems in instance migration, the quality of migration is not assessed in this experiment. We use this pair of ontologies for measuring the computational performance of the Instance Migration Engine. The transformation rules have been specified at the beginning of the experiment and remained unchanged in all stages. A stage consisted in adding an increment of the BauDataWeb to the source ABox and migrating the instances of the whole source ABox to the target ABox of the eClassOWL ontology automatically by the Instance Migration Engine. We measured the time spent for instance migration in each stage. The experiment has been run on a conventional PC workstation with the configuration given in Table 9. The experiment has been stopped after adding the 13-th increment because the memory requirements of the running tool exceeded the available RAM. The results are presented in Table 10.

Table 9. The configuration of the hardware and software for the third experiment

Hardware	
CPU	AMD Turion x2 dual-core mobile technology RM70 (2.0GHz, 1MB L2 cache)
RAM	3 GB DDR2
Operating System and Java Runtime Environment	
Runtime	Java(TM) SE Runtime Environment ver. 1.6.0_17-b04
Specification	Java Platform API Specification ver. 1.6 by Sun Microsystems Inc.
VM	Java HotSpot(TM) Client VM - mixed mode by Sun Microsystems Inc. ver. 14.3-b01
OS	Windows Vista ver. 6.0, x86

Table 10. Instance Migration run times for BauDataWeb increments

Itera-tion	No of Instances	Runtime (sec)
1	354	2
2	400	5
3	790	9
4	1289	13
5	2370	19
6	6323	26
7	9678	25
8	27013	41
9	142020	146
10	240849	253
11	849381	858
12	1607172	1753
13	2371245	Required heap size exceeds available RAM



Given the fact that the prototype Instance Migration Engine run on a conventional workstation was capable of migrating over 1.6 million assertions before exceeding the available RAM proves that the tool may be used in industrial settings. The performance

was also acceptable. As the chart shows, the run time increased almost linearly with the number of migrated instances.

8. Discussion, Conclusions, and Future Work

A number of problems caused by various reasons have been faced in the reported research and implementation work that have a common causal root – the lack of the necessary information for transforming all the relevant instances in a correct way. Instance migration problem is therefore substantially under-defined.

The provision of an explicit, correct and complete mapping of a source TBox to a target TBox is not sufficient for devising a complete and correct transformation of the corresponding assertional part. A characteristic example is the addition of a new relation to a concept in the target ontology. The values of the respective object property shall be added to the instances of the respective class. However we can not know the exact set of individuals that have to be related. Therefore such a property is not added to all instances in our approach.

A similar problem arises when the cardinality of a relation is changed. We cannot know which particular instances have to be related to the individual having the property with the changed cardinality. However it is known and could be specified in advance that the set of particular instances of a given class have to have the relation to another specific individual. Our approach allows specifying such an a priori knowledge explicitly in the transformation rules. Moreover, the log of the migration problems is collected and recorded at run time.

Another sort of migration problems arises because not all required OWL constructs are considered at proper time. A good example case is when the two classes (A and B) are defined as equivalent in the source ontology but they are not equivalent in the target ontology. Based on equivalence axioms, if the instances of A have been migrated then the instances of B have to be migrated as well. However, the removal of an equivalence axiom in the target ontology may cause collisions for particular individuals. Our approach assumes that all collisions of these sorts have to be explicitly resolved when the changes are discovered and the transformation rules are specified.

Despite that our evaluation experiments proved that the software prototype performs ontology instance migration correctly and completely if the problems mentioned above are absent or the corresponding collisions are correctly resolved in the (refined) transformation rules. The necessity of rule refinement is a good argument in favor of the iterative approach in our methodology.

Yet another kind of problems is caused by the imperfection of the transformation procedure, especially at manual steps. If a mistake is made at the step of structural change discovery it will propagate to the transformation rules and further to the automated migration. However the influence of such mistakes spans across one instance migration process iteration only. The mistakes are reported in the migration log – so the knowledge engineer may correct the rules for the subsequent iteration. Our experiments show that

even if there were some mistakes at manual steps of initial iterations, the final results still were quite good.

Another important aspect is that the evaluation metrics for correctness and completeness of the migration done by the software are not fully accurate in the sense that they result in the measurements with values being lower than it could have been estimated by manual analysis. In addition to that, the chosen metrics are not invariant to the specificity of the test data. For example, *Accuracy* increases if the number of the individuals liable to migration is considerably less than the total number of instances. *Precision* and *Recall* metrics do not provide exact reflection on migration quality. High *Precision* can be obtained at the expense of *Recall* by migrating only proper individuals but minimizing the number of migrated instances. Similarly, high *Recall* can be achieved by transferring all relevant instances that inevitably minimize *Precision.* Despite all the imperfections of the chosen metrics we have decided to use those for making the results (at least indirectly) comparable to the results of the one-shot automated ontology alignment approaches documented by OAEI 2009[23].

In a summary it may be stated that the prototype OIM tool carries out instance migration correctly based on the results of the test cases used in our first and second evaluation experiments. In terms of correctness and completeness the results are sufficiently good for attempting offering the tool for the use in industrial settings. Simplicity, transparency, and validity may be mentioned as the highlights of the developed methodology. The method and the prototype software tool allow conveniently creating, editing, and processing instance migration rules. The methodology supported by the tool also reduces the probability of errors at automated processing steps, which is proven by the high values of *Precision* measurements. The lowlights of the proposed approach are insufficient flexibility and strong correlation with the correctness of manual specification of transformation rules. These lowlights can be mitigated by offering a refined tool support for manual steps. For instance providing the means for semi-automated structural change detection in the pair of ontologies may lower the chance for manual mistakes at the preparatory steps. We have used our ODV tool for testing that. Subsequently, automated generation of transformation rules may increase the quality of the proposed approach. Both refinements of the current release of the software prototype are planned for the future work on integrating our ODV and OIM tools in one tool suite.

In distributed settings, when different ontologies formalize partial knowledge of different autonomous software agents about the same domain, a solution for ontology instance migration may facilitate more effective and efficient collaboration of those agents. Our approach is not directly applicable in these settings as it explicitly subsumes the involvement of a human knowledge engineer. However some results that have been reported in this article may be useful for implementing automated negotiations of software agents about the harmonized meaning of ontological contexts in the approaches like the one reported in Ermolayev et al.[28] We plan to investigate the applicability of our transformation patterns and rules as parts of argumentation in such negotiations among software agents in the future.

Finally it has to be highlighted that, given the limits of the currently available technologies, it is impossible to fully automate the whole process of ontology instance migration. Therefore, the reduction of the proportion of the manual work is the only feasible way of increasing the performance and quality of results in this important task of ontology engineering and management. The development of a collaborative platform for ontology engineering teams working on distributed ontologies is one more planned direction of our future work.

Acknowledgements

Cadence Design Systems GmbH retains all the rights with respect to the software mentioned in the paper, namely the ProjectNavigator and its plug-ins – ODV and OIM. The authors would like to express their gratitude to the colleagues who helped in choosing and provided the industrial ontologies for our experiments, namely to Univ.-Prof. Dr. Martin Hepp and Dipl.-Ing. Andreas Radinger, E-Business and Web Science Research Group, Universität der Bundeswehr München. The authors would also like to thank the anonymous reviewers whose comments have helped to improve the article substantially.

Reference List

1. V. Ermolayev, E. Jentzsch, N. Keberle and R. Sohnius, *Performance Simulation Initiative. The Suite of Ontologies v.2.0. Reference Specification. Technical Report PSI-ONTO-TR-2007-1*, (VCAD EMEA Cadence Design Systems, GmbH, 2007)
2. V. Ermolayev, E. Jentzsch, N. Keberle and R. Sohnius, *Performance Simulation Initiative. The Suite of Ontologies v.2.2. Reference Specification. Technical Report PSI-ONTO-TR-2007-5*, (VCAD EMEA Cadence Design Systems, GmbH, 2007)
3. V. Ermolayev, N. Keberle, W.-E. Matzke, An Upper Level Ontological Model for Engineering Design Performance Domain, LNBIP, Vol.20, (Springer Berlin Heidelberg, 2008), pp.127-141.
4. R. Sohnius, E. Jentzsch, W.-E. Matzke, Holonic Simulation of a Design System for Performance Analysis, in *Holonic and Multi-Agent Systems for Manufacturing*, Vol. 4659 (2007), (LNCS, Springer Berlin Heidelberg, 2007), pp.447-454
5. M. Hepp, Products and Services Ontologies: A Methodology for Deriving OWL Ontologies from Industrial Categorization Standards, in *Int'l Journal on Semantic Web & Information Systems*, 2(1), (2006), pp.72-99
6. O. Corcho, M. Fernández-López, A. Gómez-Pérez, Methodologies, tools and languages for building ontologies: where is their meeting point?, in *Data & Knowledge Engineering*, Vol. 46, Issue 1, (2003), pp.41 – 64
7. O. Corcho, M. Fernández-López, A. Gómez-Pérez, Ontological Engineering: Principles, Methods, Tools and Languages, in *Ontologies for Software Engineering and Software Technology*. (Springer Berlin Heidelberg, 2006), pp.1-48
8. M. Fernandez-Lopez, A. Gómez-Pérez et al, *A survey on methodologies for developing, maintaining, evaluating and reengineering ontologies*, Deliverable 1.4. (OntoWeb, 2002)
9. D. Nardi, R. J. Brachman: An Introduction to Description Logics. In: F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (eds.) *The Description Logic Handbook*, (Cambridge University Press New York, NY, USA, 2007)

10. V. Ermolayev, A. Copylov, N.Keberle, E. Jentzsch, W.-E. Matzke, Using Contexts in Ontology Structural Change Analysis, in *CEUR-WS*, Vol. 626 (2010), eds. V. Ermolayev, J.-M. Gomez-Perez, P. Haase, P. Warren, (CIAO, 2010)
11. B. Henderson-Sellers, C. Gonzalez-Perez, Standardizing Methodology Metamodelling and Notation: An ISO Exemplar, in *UNISCON 2008*, eds. R. Kaschek, C. Kop, C. Steinberger, G. Fliedl, LNBIP, Vol. 5, (Springer, Berlin/Heidelberg, 2008), pp.1-12
12. V. Vladimirov, R. Sohnius, V. Ermolayev, W.-E. Matzke, Semi-Automated Instance Migration between Evolving Ontologies, in *Special Issue: System Analysis, Management, and IT*, Herald of NTU KhPI, No 7, (2007), pp.130-144
13. M. Davidovsky, V. Ermolayev, W.-E. Matzke, V. Tolok, Evaluation of Semi-Automated Ontology Instance Migration, in *Intelligent Distributed Computing IV*, SCI 315, eds. M. Essaaidi et al., (Springer Berlin/Heidelberg 2010), pp.179-190
14. M. Davidovsky, V. Ermolayev, *Performance Simulation Initiative. Ontology Instance Migration. Report*, (Cadence Design Systems, GmbH, 2009)
15. L. Serafini, A. Tamilin Instance Migration in Heterogeneous Ontology Environments, LNCS, Vol. 4825 (2007)
16. A. Maedche, B. Motik, L. Stojanovic, Managing multiple and distributed ontologies on the SemanticWeb., VLDB, No. 12, (Springer, 2003) pp.286-302
17. M. Klein, A. Kiryakov, D. Ognyanov et al, Ontology versioning and change detection on the Web, in *Proc. of EKAW-2002*, (Siguenza, Spain, 2002) pp.197–212
18. T. Gruber, Ontolingua: A Translation Approach to Providing Portable Ontology Specifcations, in *Knowledge Acquisition*, (1993) 5(2):199-220
19. H. Chalupsky, Ontomorph: A translation system for symbolic logic, in *Proc. Int'l. Con. on Principles of Knowledge Representation and Reasoning*, (Morgan Kaufmann, San Francisco, 2000), pp.471-482
20. D. Dou, D. McDermott, P. Qi, Ontology Translation on the Semantic Web, in *The Journal on Data Semantics*, (2005)3360:35–57
21. J. Euzenat, P. Shvaiko, *Ontology Matching*, (Springer Verlag, Berlin Heidelberg, 2007)
22. J. Euzenat, A. Ferrara, Ch. Meilicke, J. Pane, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Šváb-Zamazal, V. Svátek, C. Trojahn dos Santos, Results of the Ontology Alignment Evaluation Initiative 2010, in *5th International Workshop on Ontology Matching (OM-2010)*, (2010), http://ceur-ws.org/Vol-689
23. J. Euzenat, A. Ferrara, L. Hollink, A. Isaac, C. Joslyn, V. Malaisé, Ch. Meilicke, A. Nikolov, J. Pane, M. Sabou, F. Scharffe, P. Shvaiko, V. Spiliopoulos, H. Stuckenschmidt, O. Šváb-Zamazal, V. Svátek, C. Trojahn dos Santos, G. Vouros and Sh. Wang, Results of the Ontology Alignment Evaluation Initiative 2009, in *4th International Workshop on Ontology Matching (OM-2009)*, (2009), http://ceur-ws.org/Vol-551
24. N. Fridman Noy, M. A. Musen, PROMPTDIFF: A Fixed-Point Algorithm for Comparing Ontology Versions, AAAI/IAAI, (2002), pp.744-750
25. C. Drumm et al, QuickMig - Automatic Schema Matching for Data Migration Projects, (ACM, NY, 2007), pp.107-116
26. H. H. Do, S. Melnik, E. Rahm, Comparison of Schema Matching Evaluations, LNCS, Vol. 2593, (Springer Berlin Heidelberg, 2009), pp.221-237
27. Ch. D. Manning, P. Raghavan, H. Schutze, *Introduction to Information Retrieval*, (Cambridge University Press, NY, 2008)
28. V. Ermolayev, N. Keberle, W.-E. Matzke, V. Vladimirov, A Strategy for Automated Meaning Negotiation in Distributed Information Retrieval, in *ISWC 2005 Proc. 4th Int. Semantic Web Conference*, LNCS 3729, eds. Y. Gil et al., (Springer Berlin/Heidelberg 2005), pp. 201-215