Automated Instance Migration between Evolving Ontologies

Vladimir Vladimirov[1], Richard Sohnius[2], Vadim Ermolayev[1], Wolf-Ekkehard Matzke[2]

[1]Department of IT,
Zaporozhye National University
Zhukovskogo 66, 69063, Zaporozhye, Ukraine
vvlad@zhu.edu.ua, vadim@ermolayev.com

[2]Cadence Design Systems, GmbH,
Mozart str., 2, 85622, Feldkirchen, Germany
rsohnius@cadence.com, wolf@cadence.com

Abstract: An ontology, if used for practical purposes like in information systems, contains its controlled vocabulary (TBox) describing the semantics of the domain and the set of facts (ABox) about this domain. The elements of the ABox are ontology instances. If a domain is described by two or more different ontologies or if ontologies evolve, TBoxes are the first place to analyze the differences. However, even if the corresponding TBoxes are mapped to each other, ABox alignment is still required to be done. Though TBoxes may be aligned manually, the size of the corresponding ABoxes may well be a serious obstacle for feasibly finishing the job by hand. The paper presents our approach and recent research accomplishments in developing a methodology for solving this complex task semi-automatically. We call it Ontology Instance Migration Methodology (OIMM). It allows reducing the bulk of manual work in aligning one ontology to another one. Our simplified task is to populate the second one with the instances taken from the first one. We first build mappings between the TBoxes, then we proceed with creating an Ontology Instance Migration Scenario (OIMS) using the algorithm presented in this paper and the previously created mappings. We review the OIMS to validate it, edit incomplete and add missed transformations. Such manual additions are necessary to encode complex transformations. We finally execute the OIMS using the environment which performs the instance migration.

1 Introduction

Instance migrations are the part of the ontology development and ontology alignment activities. The ontology development process often includes an evolution of the ontology design. It is common practice to change the TBox ontology part during the ontology development cycle, but the set of facts remain the same and needs to be encoded according to the new TBox. Ontology alignment is used to allow interoperability between heterogeneous data sources described with their own ontologies. In both cases, the TBox is the first place to analyze the differences when performing alignment. It is also possible to map pairs of TBox-es to each other – this can be done for example manually. But the size of the corresponding ABox makes manual migration of the instances a complicated task. We describe an approach and an algorithm for solving this

complex task semi-automatically. The presented approach helps to create instance migration scenario and thus gives the user full control of the instance migration process by changing scenario blocks and their order. Users will be able to develop scenarios using the provided programming library facilities and run this scenario as a program in an execution environment. TBox mappings will be used to map source ontology instance property values to target instance property values.

The Performance Simulation Initiative (PSI) project was started by Cadence Design Systems GmbH to help customers in finding and improving the weak spots in their design processes and thereby increasing performance. In the PSI project we acquire and formalize data about dynamic engineering design processes in microelectronics design projects by acquiring knowledge about them using our in-house methodology [So06] based on PSI family of OWL ontologies [Er06a]. Data from one of the real projects run at Cadence was used as test case data to evaluate the PSI family ontologies and was encoded as ABox ontology part – the data was represented as OWL instances. The PSI family ontologies evolved since the project start from version 1.0 to 1.6. Changes were mainly made in the TBox ontology part. To perform tests, the test case data needed to be reformatted using the most current ontology version.

The paper is structured as follows. Section 2 specifies initial requirements and the ancestor solution used to solve a similar task. Section 3 presents the Ontology Instance Migration Methodology. Section 4 explains the methodology steps in detail using an example. Section 5 summarizes the results and contains the plans for the future work.

2 Current solution - instance population script

The ancestor of the presented methodology is the instance population script used in the PSI project as current solution. The Instance Population Program (IPP) was developed to automate the task of encoding the test case project data in OWL. Perl[1] was chosen as implementation language.

As input data IPP accepts tables with related items which have to be stored in CSV[2] text files. The CSV format has not the ability to store relations between items so these relations' information was encoded inside the IPP. The IPP generates OWL instances for the project data according to the TBox and saves them into OWL files.

It performs instance population with test case project data loaded from .csv text files by creating instances for ontology classes, setting the property values for them, and defining object relations by setting object properties. The property value and object property definition process is partially hardcoded in the program modules. Thus a new IPP version had to be developed whenever a new PSI ontology family version was designed.

---

[1] Perl (Practical Extraction and Report Language) is a dynamic programming language. Perl borrows features from a variety of other languages including C, shell scripting (sh), AWK, sed and Lisp.
[2] RFC4180. Common Format and MIME Type for Comma-Separated Values (CSV) Files http://tools.ietf.org/html/rfc4180

IPP performs the following actions:

1) Loads the TBox: owl:import links, Name Spaces, concept names, properties and URIs, and processes the imported ontologies

2) Reads the data from csv text files and uses it to create OWL instances, set property values, and create links with other instances

3) Performs a set of instance data validation tests

4) Writes out the OWL-file(s)

The amount of work to develop a new IPP version was much less in comparison to the amount of work to develop the first IPP version from scratch. Initially it was necessary to update the TBox encoding for each new ontology version while the other IPP parts were left mostly unchanged. Later versions of the IPP were also able to load the TBox completely from OWL-files and manual changes were then only done to the instance generation part.

Currently, all instance populations for new ontology versions in PSI are performed using the IPP. So far 3 major versions were developed (to populate PSI ontologies from v.1.4 to v.1.6 with test case instances). Additionally, about 10 minor versions have been written during ontology development and refinement. The latest version allows populating PSI Ontologies Suite v.1.6 [Er06b] with the instances corresponding to 4 different test cases. The IPP helps to generate roughly 400 instances, fill their property values and establish object relations between the generated instances.

It reduces the amount of work to generate ABox data from several test case data sets for a predefined TBox hard-wired in the IPP script. It also allows output ontology TBox changes with little amount of work on updating the program code. It took weeks to develop the IPP, but with it version updates are done in about 30 minutes.

The IPP is a specific purpose tool and as such it cannot work with ontologies different from the one it has been developed for. To make it work with other ontologies, the program code has to be seriously revised manually. The complexity of this revision is comparable with the complexity of developing it from scratch. Furthermore, the IPP cannot be used without substantial reprogramming for different output ontologies and source data.

3 Ontology Instance Migration Methodology

The main objective of developing our OIMM was to provide a feasible way of actually migrating instances between ontology versions instead of rebuilding the instances from scratch. The most important point was to reduce the work on defining instance data transformation to a minimum. The methodology should allow an ontology engineer to

spend less time on creating instances for new ontology versions. It should also be usable without the substantial programming knowledge needed for updating the IPP.

To automate loading the source information and storing the resulting information, we use the Jena API[3] which supports parsing ontology files into an ontology model, creates objects for concepts and instances within this model and allows to save the ontology model to a file. To define the instance data transformation, we propose to create mappings between the TBoxes of ontologies using existing tools with convenient graphical user interfaces.

To populate the new ontology with the instances taken from the old one, we present a methodology which consists of the following steps: (i) build mappings between the TBoxes; (ii) create an Ontology Instance Migration Scenario (OIMS) using these mappings and algorithm patterns; (iii) execute the OIMS to perform instance migration. The OntoStudio with the OntoMap[We05] plugin is used to create the ontology mappings of step 1, the scenario of step 2 is build by manually extending algorithms (section 3.2). In step 3, we need to execute the scenario encoded as a Python script. For that, we are developing a scenario execution environment. The execution environment requirements are presented in section 3.3.

3.1 Mapping Creation

An ontology as a knowledge base consists of conceptual or terminological knowledge (TBox) and instance or assertional knowledge (ABox). The TBox is represented by sets of the following ontology entities - concepts, concept properties and concept relations. In the following, we shall call a single correspondence between a set of ontological entities a *mapping rule* and a set of mapping rules between a source and a target ontology a *mapping*. A simple mapping rule is a one-to-one concept relation. Complex mapping rules define one-to-many, many-to-one, many-to-many relationships.

In our work to create and manage mappings in a graphical environment, we have chosen the OntoMap plugin for the ontology-management platform OntoStudio . It supports the creation and management of ontology mappings via a graphical interface. Mappings can be specified based on graphical representation, using a schema-view of the respective ontologies. Users just have to understand the semantics of the graphical representation (e.g. an arrow connecting two concepts), they do not have to worry about the logical representation mappings. The user of OntoMap is supported by drag-and-drop functionality and simple consistency checks on property-mappings (automatic suggestion of necessary class-mappings). For concept mappings constraints can be specified on the available attributes, based on a form.

OntoStudio has its own grounding of mappings, based on F-Logic rules [Ki97].

---

[3] Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. Jena is open source and grown out of work with the HP Labs Semantic Web Programme. http://jena.sourceforge.net/

OntoMap supports simple types of mappings. If more complex mappings are needed (possibly using complex logical expressions or built-ins), they have to be encoded manually. Despite that, OntoMap covers a substantial number of use-cases. The rules that are currently supported include: concept to concept mappings, attribute to attribute mappings, relation to relation mappings, attribute to concept mappings [We05].

The language for ontology mappings used in OntoMap has been developed as language-neutral representation of mappings that correspond to mapping-patterns [Br05a]. Mapping patterns represent a schema for frequent mappings. Persistency of mappings defined in the Mapping Language is supported by a Language API Java-library [Sch05]. It is possible to export mappings from OntoMap to the OWL format. Mappings in this format can be parsed and reused.

The first OIMM step is to create mappings with the OntoMap tool by connecting related concepts, properties and relations. To specify complex concept mappings two or more one-to-one simple concept mappings need to be created. Then the mapping rules are exported to an OWL file. These mappings will be used in the Ontology Instance Migration Scenario.

3.2 Creating Ontology Instance Migration Scenario

An Ontology Instance Migration Scenario can be described as an IPP extended with the capability to accept input data as an arbitrary OWL ontology and a predefined set of actions for instance creation and linking using mapping rules.

The OIMS algorithm consists of blocks similar to the ones of an IPP:

1) Load information about the TBox of the source ontology *Os* and the target ontology *Ot*: owl:import links, Name Spaces, concept names, properties and URIs. Load imported ontologies. Load ABox instance data of the *Os*.

2) Populate instances for *Ot*, setting property values and creating object property links with other instances using the TBox and the ABox of the *Os* and the defined mapping rules. Simple concept mappings and property mappings will be used to set *Ot* instance property values with *Os* instance property values mapped to them;

The following steps need be performed for instance population:

(i) Create target instances and set property values. These newly made instances are not linked with each other with object properties on this step. Also prepare list of the source ontology concepts not mapped with concepts in target ontology.

```
prepare o1sorted - the list of concepts from O1 sorted by the number of corresponding
instances ascending
create ABox2 ontology model
foreach Cs in o1sorted do
  if exists GetMR(C) then
    if GetMR(Cs).mappingtype is one-to-one or many-to-one:
```

```
      Ct = GetMR(Cs).target
      foreach Is in instances of Cs:
        create instance It of Ct in the ABox2 ontology model
        set It.source_instance=Is
        set It.ccmapping_rule=MR(Cs)
        set It.source_concept=Cs
        setvalues(It,Is,MR(Cs).propertymapping)
    else
      add C to the ComplexMappedConceptsList
  else
    add C to the NotMappedConceptsList
```

GetMR(C) – get concept mapping rule for concept C (as a source). **setvalues(It,Is,MR(Cs).propertymapping)** – set property values of instance It with property values from the Is, converting them using property mappings for concept Cs.

(ii) The user reviews list of not mapped concepts and if some mappings were missing, adds them using the mapping editor tool and performs step 1 again. Or he(she) manually adds commands for instance creation. The user should provide the following information in his commands: source concept(s), target concepts, and rules to create and fill instances using source instances. This info will be used on step 3 to add instance relations.

```
foreach Is in instances of Cs:
  create instance It of Ct in the ABox2 ontology model
  set It.source_instance=Is
  set It.ccmapping_rule='manual'
  set It.source_concept=Cs
  customsetvalues(It,Is)
```

customsetvalues(It,Is) – custom function which defines a manually encoded instance to instance mapping algorithm. To help an ontology engineer to create custom mapping functions, we are planning to develop patterns of custom instance to instance mapping algorithms for different ontology mapping cases. Then we shall collect these patterns in a library.

(iii) Instance relations creation is performed by the following algorithm: Search Ot for instances to be related to one another and find the objectproperty to be set to relate the instances. It also detects the concepts without instances from the target ontology and add then to the list for manual inspection.

```
foreach Ct in O2ConceptList:
  IList = I(Ct)
  foreach or in GetObjectPropertiesFor(Ct):
    Cm = or.getRDFSRange
    IMappedList = I(Cm)
    foreach inst in IList:
      sourceinst = inst.source_instance
      SourceConnectedList = GetSourceConnectedList(sourceinst)
      if size(SourceConnectedList)>0 then
        for scinst in SourceConnectedList:
          for icm in I(Cm):
            if icm.source_instance==scinst
```

```
                relate(inst,icm,or)
foreach Ct in O2ConceptList:
  if I(Ct)==0 then
    add Ct to the ConceptsWithoutInstances list
```

GetObjectPropertiesFor(C) – get list of concept's object properties.
I(C) – get instances list for the Concept
GetSourceConnectedList(sourceinst) – get the list of instances connected with source instance.
Size(L) –get array size

(iv) The user checks concepts from the ConceptsWithoutInstances list and adds commands for creating instances if it is required.

3) Perform instance data validation tests to check restrictions defined in *Ot*;

4) Write out results to OWL-file(s).

3.3 Scenario execution environment requirements

Execution of blocks 1, 3, 4 can be easily encoded as a program. User should manually define the part of block 2 to edit incomplete and add missed transformations. A scenario can be encoded as Python[4] script. Already formalized methods will be implemented as programming library. The execution environment will consist of an interpreter for the programming language and a common function library.

Initial requirements for the execution environment are the following:

Programming language requirements – A language with syntax close to natural language; interpreted scripting language – so user will not have to manually compile program on every update; should have the ability to reuse existing Java ontology management libraries. Using Jython[5] will help to reuse Java libs from scripts written on Python.

The common function library should contain implementation of the following actions performed within the described methodology: Load source ontology TBox and ABox, load target ontology TBox, create instance of concept, iterate over ontology concepts, iterate over instances for concept or related to another instance, access concept and instance property values and relations, select instances of class, find mapping by source concept, check concept mapping type.

---

[4] Python is an interpreted, object-oriented programming language, http://www.python.org/
[5] Jython is an Java implementation of the high-level, dynamic, object-oriented language Python, and integrated with the Java platform. It allows to run Python on any Java platform and to use existing Java libraries from Python,http://www.jython.org/Project/index.html

4 Mapping instances evaluation example

To illustrate the presented methodology we shall use a fragment of the evolving Task-Activity (TA) ontology of the PSI ontologies suite [Er06a, Er06b].

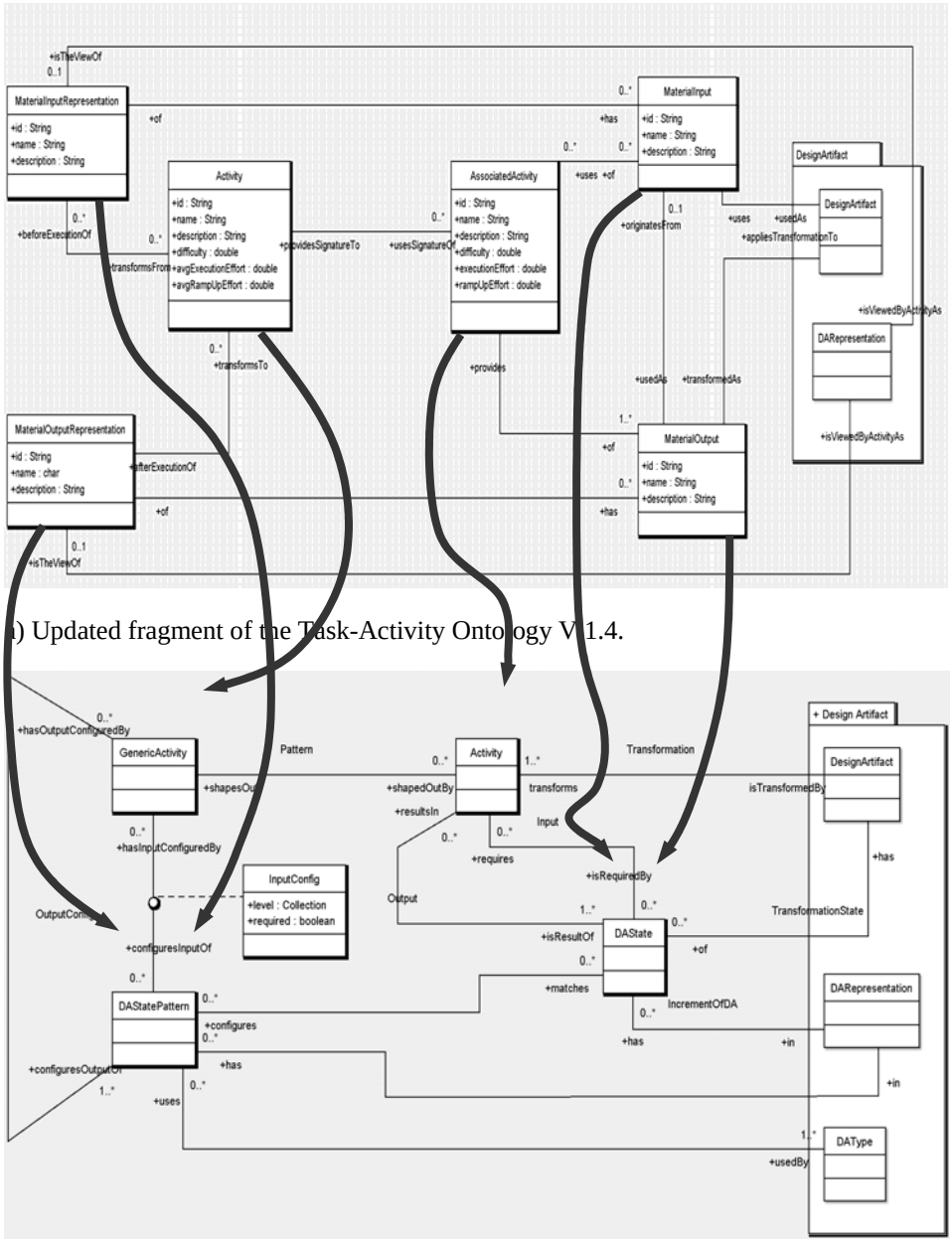
In this example the fragment of TA evolves from v.1.4 [Er06] to v.1.5 [Er06a] and the following pieces are available: TBox and ABox of TA ontology version 1.4 and TBox only for ontology version 1.5. We also have the descriptions of the changes in the ontology specification [Er06a]. The task is to migrate the instances from the ABox of TA ontology version 1.4 into the ABox of TA ontology version 1.5.

The fragment of the Task-Activity Ontology V.1.4 to which the changes are applied is shown in Fig. 1a. The changes in the Task-Activity Ontology are shown in Fig. 1b. Fig 1 also shows the substitutions of the concepts.

MaterialInputRepresentation, **MaterialOutputRepresentation**, **MaterialInput**, and **MaterialOutput** concepts were dismissed and replaced by the concepts of **DAStatePattern** for a **GenericActivity** and the concept of a **DAState** for an **Activity**. The **Activity** concept was renamed to **GenericActivity**. **The InputConfig** concept was added. The properties of an **InputConfig** association concept are: **level** – Collection: denotes the location of the intended source (DesignArtifact) of an input in the decomposition hierarchy, and **required**: Boolean – describes that at least one matching DA must be found or the activity cannot be applied. **GenericActivity** receives a new property: **effortLimit**: double – the upper bound value of the effort spent to execute the described activity. The **AssociatedActivity** concept was renamed to **Activity.**

The example was chosen because: (i) it is compact enough to be presented in a paper; (ii) the fragment contains complex transformations where instances from two or many concepts should be transferred into the instances of one concept (many-to-one complex concept mapping).

At first, we build mappings between the ontology TBox parts of these versions using the OntoMap plugin (Fig 2). After creating the mappings we export then into an .OWL file (Fig. 3).



a) Updated fragment of the Task-Activity Ontology V 1.4.

Fig. 1: The changes in Task-Activity Ontology.

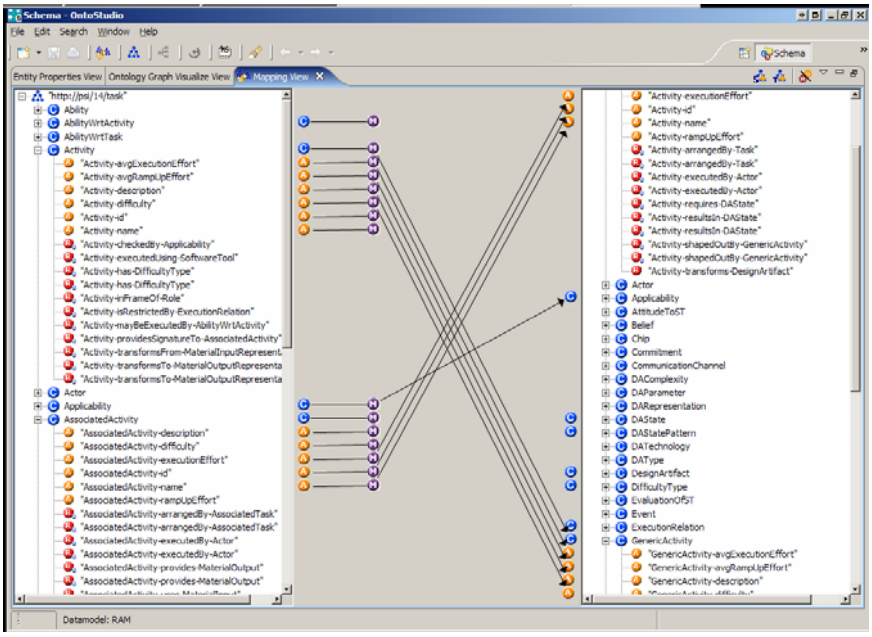


Fig 2. Concept and property mappings created in the OntoMap

```
<owl:Thing
rdf:about="mapping_source_("http://psi/15/task"#CCMap_Mater
ialOutputRepresentation_DAStatePattern,
"http://psi/14/task"#MaterialOutputRepresentation)">
<b:type_>Concept</b:type_>
<b:sourceTransformation_>""</b:sourceTransformation_>
<b:sourceModule_>"http://psi/14/task"</b:sourceModule_>
<b:sourceId_
rdf:resource="http://psi/14/task#MaterialOutputRepresentation"
/>
</owl:Thing>
<owl:Thing
rdf:about="mapping_source_("http://psi/15/task"#CCMap_Mater
ialInputRepresentation_DAStatePattern,
"http://psi/14/task"#MaterialInputRepresentation)">
<b:type_>Concept</b:type_><b:sourceTransformation_>""</b
:sourceTransformation_><b:sourceModule_>"http://psi/14/task"
</b:sourceModule_>
<b:sourceId_rdf:resource="http://psi/14/task#MaterialInputRepr
esentation" />
</owl:Thing>

<owl:Thing
rdf:about="mapping_source_("http://psi/15/task"#AAMapping11
71428954501,"http://psi/14/task"#"AssociatedActivity-name")">
<b:type_>Attribute</b:type_>
<b:sourceRange_
rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
<b:sourceConcept_
rdf:resource="http://psi/14/task#AssociatedActivity" />
<b:sourceTransformation_>""</b:sourceTransformation_>
<b:sourceModule_>"http://psi/14/task"</b:sourceModule_>
<b:sourceId_
rdf:resource="http://psi/14/task#"AssociatedActivity-name"" />
</owl:Thing>
```

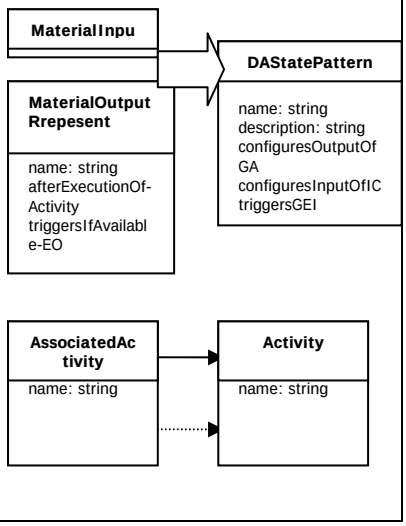


Fig 3. Mappings presented using Ontology Mapping Specifiation Language encoded in OWL format

The following mapping rule set fragment will be used in our example:

Mapping rule id	Source	target
CCMapping1171829754406	http://psi/14/task#Precondition	http://psi/15/task#Precondition
CCMapping1171829427390	http://psi/14/task#MaterialOutputRepresentation	http://psi/15/task#DAStatePattern
CCMapping1171829425656	http://psi/14/task#MaterialInputRepresentation	http://psi/15/task#DAStatePattern

Mapping rules CCMapping1171829425656 and CCMapping1171829427390 have the same target concept http://psi/15/task#DAStatePattern which means that this is a many-to-one concept mapping rule.

While performing step 1 of the instance population scenario we create new instances. The concept MaterialOutputRepresentation in the source ontology has 13 instances: MaterialOutputRepresentation_1 .. MaterialOutputRepresentation_13. According to the algorithm, we should create a new instance in the target ontology for each of them. The source ontology instance MaterialOutputRepresentation_3 has the following properties:

MaterialOutputRepresentation_3 properties:
MaterialOutputRepresentation-triggersIfAvailable-EventualOutput = http://psi/14/ssmhc#EventualOutput_11
MaterialOutputRepresentation-isTheViewOf-DARepresentation = http://psi/14/ssmhc#DARepresentation_3
MaterialOutputRepresentation-afterExecutionOf-Activity = http://psi/14/ssmhc#Activity_13
MaterialOutputRepresentation-name=Testplan
rdf#type =http://psi/14/task#MaterialOutputRepresentation

On the target side, we create an empty instance http://psi/15/ssmhc#DAStatePattern_1. The MaterialOutputRepresentation's name property is equal to the DAStatePattern's name property. Therefore, we set the value of the name property for the new instance with corresponding values from the source instance. The Id is generated automatically with the internal instance counter. We also set algorithm specific values:

```
# DAStatePattern-name=Testplan
# DAStatePattern-id=1
#_source_concept= http://psi/14/task#MaterialOutputRepresentation_
#_source_instance= http://psi/14/task#MaterialOutputRepresentation_3
```

Then we perform similar actions for all concepts' instances from the source ontology.

After creating all instances in the target ontology we need to establish the object relations between them. Lets do this for the instance http://psi/15/ssmhc#DAStatePattern_1 created from MaterialOutputRepresentation_3 in the example before.

After step 1, the concept http://psi/15/task#DAStatePattern has 26 instances which were created from 12 MaterialOutputRepresentation instances and 14 MaterialInputRepresentation instances according to many-to-one concept mapping rule.

The concept DAStatePattern has the following object properties: DAStatePattern-configuresOutputOf-GenericActivity, DAStatePattern-configures-DAState, DAStatePattern-configuresInputOf-InputConfig, DAStatePattern-triggers-GenericEventualInput, DAStatePattern-triggersIfAvailable-GenericEventualOutput. For each object property, we get its range concept from the object property definition. For example for the DAStatePattern-configuresOutputOf-GenericActivity it will be the GenericActivity concept.

For the GenericActivity concept we have 13 instances created from instances of the Activity. We will iterate over GenericActivity instances getting _source_instance. For GenericActivity_13, the source instance in our example is Activity_13. It is linked with the MaterialOutputRepresentation_3 instance by the following object property: Activity-transformsTo-MaterialOutputRepresentation = MaterialOutputRepresentation_3.

So one of the instances related with Activity_13 instance from the source ontology is MaterialOutputRepresentation_3 and it matches the source instance of DAStatePattern. Therefoe, we can set the object property value GenericActivity-hasOutputConfiguredBy-DAStatePattern of the instance GenericActivity_13 with the instance DAStatePattern_1.

During the instance migration for this example, we have been able to create and link instances for 11 concepts using the presented algorithm. Manual commands were required to create instances of one concept – InputConfig.

5 Discussion and Related Work

Our methodology connects 3 core components: mapping language, graphical user interface to build mapping rules, and an instance transformation tool. There are other ontology alignment solutions which cover all or part of these three components. They are WSMO Studio[Si06] and WSMX[Ci05] tools developed as a part of DIP project[Ha04], OntoMap Ontoprise plugin[We05] developed as a part of SEKT[6] project, and PromptMap Protégé editor plugin[7] from Protégé team.

OntoPrise OntoStudio OntoMap plugin provide the most convenient GUI to build mapping rules in comparison with the others. But they do not have a tool to perform instance transformation yet. PromptMap and WSMX have instance mapping facilities controlled by mapping rules. But after creating mapping rules, the user is not able to control the instance transformation process. Our methodology allows doing this.

---

[6] Semantic Knowledge Technologies (SEKT) developed and exploited semantic knowledge technologies. Core to the SEKT project has been the creation of synergies by combining the three core research areas ontology management, machine learning and natural language processing. http://www.sekt-project.org/

[7] The PROMPT plug-in allows to manage multiple ontologies in Protégé Ontology Editor, http://protege.cim3.net/cgi-bin/wiki.pl?Prompt

	WSMO Studio	Protégé Mapping Tab	Ontoprise OntoStudio
GUI functionality:			
One-to-one concept mappings building support	yes	yes	yes
One-to-many, many-to-many concept mappings	yes	no	yes
Property mappings	no	yes	yes
objectproperty mappings	yes	no	yes
Importing from owl files support	yes	yes	yes*
Exporting to the owl format support	no	yes	yes
Used mapping language	Mapping Language[SB05]	An Ontology of Mapping Relations by Monica Crubezy team, Stanford University [Cr03]	Mapping Language[SB05]

* The current version of the Ontoprise OntoStudio (Version: 1.6.0 Build id: 1003) does not load owl ontologies distributed in more then one file and connected by owl:imports statement. In our example we have to glue PSI ontology files into one.

The methodology we are presenting in this paper uses the existing components: mappings representation language[Sb05] and OntoMap as GUI mapping rules builder. And concentrates on developing its own instance transformation tool. Our approach should allow an ontology engineer to spend less time on performing instance transformation between ontologies by using mapping rules to generate instance transformation scenario. Scenario is encoded as program thus gives the user full control on it by editing this program.

In our methodology, the instance transformation is performed within the scenario execution environment. The execution environment consists of an interpreter for the programming language and a common function library. We reviewed several programming languages (see following list) to choose a programming language and have

chosen Jython as the most developed Java integration solution in comparison with the others.

	Java	Perl	Python	Ruby
Is there need to learn new language	No	No, Perl is already used in PSI	Yes	Yes
Possible to reuse Java code	Yes	No	Yes, Jython	Yes, JRuby
Need to be compiled	Yes	No	No	No
Has owl ontology management libs	Yes	Yes	Yes	Yes
Full integration with Java lib version	-	- (existing solution requires compilation of program)	Jython 2.2 Beta1, JPype 0.5.2.1	JRuby 0.9.2
Easy to learn	No	Yes	Yes	Yes

Concluding Remarks

This paper presents our results in developing the Ontology Instance Migration Methodology. The main goal of this methodology is to populate an ontology with the instances taken from another one using a TBox mapping and thus reducing the volume of manual work in aligning one ontology to the other one. We used existing ontology mapping methods and an algorithm based on initial instance population program used in the PSI project to develop a general OIMM which provides the ability to perform migrations between a pair of ontologies. In this paper, we presented the algorithm for an ontology instances migration scenario and gathered the requirements for its implementation.

In the future, we are going to implement the scenario execution environment according to specified requirements and develop a programming library for the implementation of scenario algorithms. It will provide ready to use commands and programming blocks for all typical instance migration tasks. We are also planning to develop more custom instance population algorithms for other ontology fragments and gather them as an algorithm pattern library. The next step will be an automatic generation of instance transformation scenarios based on TBox mappings.

Acknowledgements

We would like to acknowledge the substantial aid by all the members of the PSI project team in discussing the instance migration methodology and working on the initial requirements.

Bibliography

[So06] **Sohnius, R.** et. al.: Managing Concurrent Engineering Design Processes and Associated Knowledge. In: Ghodous, P., Dieng-Kuntz, R., and Loureiro, G. (Eds.): Leading the Web in Concurrent Engineering. Proc. 13th ISPE Int Conf on Concurrent Engineering: Research and Applications, 18 - 22 Sept., Antibes, French Riviera, IOS Press, Series: Frontiers in AI and Applications, Vol. 143, pp. 198-205, 2006

[SB05] **Scharffe F.**; **de Bruijn J.**: A Language to specify Mappings between Ontologies, IEEE Conference on Internet-Based Systems (SITIS6), December 2005, Yaounde, Cameroon.

[Sch05] **Scharffe F.**: Ontology Mapping Specification Language API. Prototype Fact Sheet, 2005, http://www.omwg.org/tools/omaplang/v0.2/FactSheet.html

[Br05] **de Bruijn J. et. al.**: The Web Service Modeling Language WSML. WSML Final Draft 5 October 2005, http://www.wsmo.org/TR/d16/d16.1/v0.21/

[Br05a] **de Bruijn, J.; Foxvog, D.; Zimmerman, K.:** Ontology Mediation Patterns Library V1. SEKT Project Deliverable D4.3.1. 2005

[Er06] **Ermolayev, V. et. al.**: The Family of PSI Ontologies Version 1.4. Reference Specification. Technical Report. Cadence Design Systems, GmbH, 47 p., 2006

[Er06a] **Ermolayev. V. et. al.**: Performance Simulation Initiative. The Family of PSI Ontologies v.1.5. Reference Specification. Technical Report PSI-ONTO-TR-2006-3, 14.04.2006, Cadence Design Systems, GmbH, 56 p.

[Er06b] **Ermolayev. V. et. al.**: Performance Simulation Initiative. The Family of PSI Ontologies v.1.6. Reference Specification., 2006, Cadence Design Systems, GmbH.

[We05] **Weiten M., Wenke D., Meier-Collin. M.**: D4.5.3 Prototype of the ontology mediation software V1, 2005

[KL97] **Kifer, M. And Lausen, G.**: FLogic: A higher-order language for reasoning about objects. SIGMOD Record, Vol. 18 (1997) No. 6, pp 134-146

[MH04] **McGuinness D., van Harmelen F.**: OWL Web Ontology Language Overview, 2004, http://www.w3.org/TR/owl-features/

[Cr03] **Crubezy M., Pincus Z., Musen M.**: Mediating Knowledge between Application Components, SMI-2003-0978, 2003

[Si06] Simov, A. et. al.: D4.11: WSMO Studio v2, DIP Project, WP 4b 2006, WSMO Platform & Tools. http://dip.semanticweb.org

[Ci05] **Cimpian, E.; Vitvar, T.; Zaremba, M.**: "D13.0v0.2 Overview and Scope of WSMX", WSMX Working Draft 2005-02-23. http://www.wsmo.org/TR/d13/d13.0/v0.2/

[Ha04] **Hauswirth, M. et al.**: D6.2: DIP Architecture, DIP Project, WP6 Interoperability and Architecture. 2004. http://dip.semanticweb.org.