DIP

Data, Information and Process Integration with Semantic Web Services

FP6 - 507483

Deliverable

WP 2: Ontology Management
D2.6
Mapping Tool Background v3

François Scharffe

June 30th, 2006
EXECUTIVE SUMMARY

This document comes with deliverable 2.6 to present the necessary background to build the DIP ontology mapping tool. We particularly express a set of requirements that such a tool must fulfil, and give refinements to the mapping language that was first presented in the version 1 and 2 of this document. The API dealing with the mapping language on top of which the DIP Ontology Mapping Tool is built has been refined and extended to follow the changes of the language.

Ontology mediation is the key technology to enable semantically described web services. By connecting domain ontologies used to described the services, ontology mediation enable the communication that makes web services automatically interoperable. Ontology mediation acts at many level of the Dip project, as an essential component of web services data mediation. In that respect the presented work is strongly related to the work package 5 of the project.

This document present the necessary background to enable ontology mediation, focusing on the design phase of the mediation process. Ontology mappings are the bridges that relate different ontologies. Realizing these mappings is a time consuming task that we propose to dramatically shorten, using the mapping tool implemented on the basis of the results presented in this document.

We are giving theoretical results with a strong emphasis on their technical use, in order to assist the construction of a mapping tool. We present requirements that may help mapping tools developers to construct efficient products. This document is part of the ontology management suite as proposed in the context of this project.

Disclaimer: The DIP Consortium is proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.
### Document Information

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>FP6 – 507483</th>
<th>Acronym</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full title</td>
<td>Data, Information, and Process Integration with Semantic Web Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project URL</td>
<td><a href="http://dip.semanticweb.org">http://dip.semanticweb.org</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Document URL</td>
<td>Kai Tullius</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>document</th>
<th>Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work package</td>
<td>Number</td>
<td>Title</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Ontology Management</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date of delivery</th>
<th>Contractual</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>M 24</td>
<td>30-Jan-06</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Version. 2.2</th>
<th>final X</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Nature</th>
<th>Prototype</th>
<th>Report x Dissemination</th>
<th>Ontology</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Dissemination Level</th>
<th>Public X</th>
<th>Consortium</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Authors (Partner)</th>
<th>François Scharffe (UIBK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsible Author</td>
<td>François Scharffe</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:Francois.scharffe@deri.org">Francois.scharffe@deri.org</a></td>
</tr>
<tr>
<td>Partner</td>
<td>UIBK</td>
</tr>
<tr>
<td>Phone</td>
<td>0043-512-507-6448</td>
</tr>
</tbody>
</table>

### Abstract (for dissemination)

The document presents requirements elaborated for an ontology mapping tool.

### Keywords

Ontology, mapping, mediation, alignment, data, tool, requirements

### Version Log

<table>
<thead>
<tr>
<th>Issue Date &lt;dd-mmm-yy&gt;</th>
<th>Rev No.</th>
<th>Author &lt;author name&gt;</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>19-Jan-06</td>
<td>2.00</td>
<td>François Scharffe</td>
<td>First Draft</td>
</tr>
<tr>
<td>26-Jan-06</td>
<td>2.1</td>
<td>François Scharffe</td>
<td>Reviewer comments</td>
</tr>
<tr>
<td>30-Jan-06</td>
<td>2.2</td>
<td>François Scharffe</td>
<td>Reviewer comments + new template</td>
</tr>
<tr>
<td>30-Jun-06</td>
<td>2.3</td>
<td>François Scharffe</td>
<td>Minor changes and adaptation to delivery template</td>
</tr>
<tr>
<td>Reviewer</td>
<td>Email</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adrian Mocan</td>
<td><a href="mailto:Adrian.mocan@deri.org">Adrian.mocan@deri.org</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partner</td>
<td>NUIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phone</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Ireland Galway                                  | NUIG    | Dr. Sigurd Harand
Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway
Galway
Ireland
Email: sigurd.harand@deri.org
Tel: +353 91 495112 |
| Fundacion De La Innovacion.Bankinter                                   | Bankinter | Monica Martinez Montes
Fundacion de la Innovation. Bankinter
Paseo Castellana, 29
28046 Madrid, Spain
Email: mmtnez@bankinter.es
Tel: 916234238 |
| British Telecommunications Plc.                                        | BT      | Dr John Davies
BT Exact (Orion Floor 5 pp12)
Adastral Park Martlesham
Ipswich IP5 3RE,
United Kingdom
Email: john.nj.davies@bt.com
Tel: +44 1473 609583 |
| Swiss Federal Institute of Technology, Lausanne                       | EPFL    | Prof. Karl Aberer
Distributed Information Systems Laboratory
École Polytechnique Fédérale de Lausanne
Bât. PSE-A
1015 Lausanne, Switzerland
Email: Karl.Aberer@epfl.ch
Tel: +41 21 693 4679 |
| Essex County Council                                                   | Essex   | Mary Rowlett,
Essex County Council
PO Box 11, County Hall, Duke Street
Chelmsford, Essex, CM1 1LX
United Kingdom.
Email: maryr@essexcc.gov.uk
Tel: +44 (0)1245 436524 |
| Forschungszentrum Informatik                                           | FZI     | Andreas Abecker
Forschungszentrum Informatik
Haid-und-Neu Strasse 10-14
76131 Karlsruhe
Germany
Email: abecker@fzi.de
Tel: +49 721 9654 0 |
| Institut für Informatik, Leopold-Franzens Universität Innsbruck       | UIBK    | Prof. Dieter Fensel
Institute of computer science
University of Innsbruck
Technikerstr. 25
A-6020 Innsbruck, Austria
Email: dieter.fensel@deri.org
Tel: +43 512 5076485 |
<table>
<thead>
<tr>
<th>Company</th>
<th>Contact Name</th>
<th>Address</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILOG SA</td>
<td>Christian de Sainte Marie</td>
<td>9 Rue de Verdun, 94253 Gentilly, France</td>
<td><a href="mailto:csma@ilog.fr">csma@ilog.fr</a></td>
<td>+33 1 49082981</td>
</tr>
<tr>
<td>inubit AG</td>
<td>Torsten Schmale</td>
<td>inubit AG Lützowstraße 105-106 D-10785 Berlin</td>
<td><a href="mailto:ts@inubit.com">ts@inubit.com</a></td>
<td>+49 30726112 0</td>
</tr>
<tr>
<td>Intelligent Software Components, S.A.</td>
<td>Dr. V. Richard Benjamins, Director R&amp;D</td>
<td>Pedro de Valdivia 10 28006 Madrid, Spain</td>
<td><a href="mailto:rbenjamins@isoco.com">rbenjamins@isoco.com</a></td>
<td>+34 913 349 797</td>
</tr>
<tr>
<td>MDR Partners</td>
<td>Rob Davies</td>
<td>MDR Partners 8 St. Andrew Street Hertford, Herts. United Kingdom, SG14 1JA,</td>
<td><a href="mailto:rob.davies@mdrpartners.com">rob.davies@mdrpartners.com</a></td>
<td>+44 (0)208 8763121</td>
</tr>
<tr>
<td>Hanival Internet Services GmbH</td>
<td>Alexander Wahler</td>
<td>Hanival Internet Services GmbH Kirchensasse 13/1a A-1070 Wien</td>
<td><a href="mailto:wahler@niwa.at">wahler@niwa.at</a></td>
<td>+43 (0)1 3195843-11</td>
</tr>
<tr>
<td>The Open University</td>
<td>Dr. John Domingue</td>
<td>Knowledge Media Institute The Open University, Walton Hall Milton Keynes, MK7 6AA United Kingdom</td>
<td><a href="mailto:j.b.domingue@open.ac.uk">j.b.domingue@open.ac.uk</a></td>
<td>+44 1908 655014</td>
</tr>
<tr>
<td>SAP AG</td>
<td>Dr. Elmar Dorner</td>
<td>SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany</td>
<td><a href="mailto:elmar.dorner@sap.com">elmar.dorner@sap.com</a></td>
<td>+49 721 6902 31</td>
</tr>
<tr>
<td>Sirma Al Ltd.</td>
<td>Atanas Kiryakov</td>
<td>Sirma Ontotext Lab, - Sirma Al EAD Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse Sofia 1784, Bulgaria</td>
<td><a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a></td>
<td>+359 2 9768 303</td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Jeff Eisenberg</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>VUB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Unicorn Solutions Ltd, Malcha Technology Park 1, Jerusalem 96951, Israel
Email: jeff.eisenberg@unicorn.com
Tel.: +972 2 6491111 | Pieter De Leenheer
Starlab- VUB
Vrije Universiteit Brussel
Pleinlaan 2, G-10
1050 Brussel, Belgium
Email: Pieter.De.Leenheer@vub.ac.be
Tel.: +32 (0) 2 629 3749 |
LIST OF KEY WORDS/ABBREVIATIONS

Ontology mapping: Relating overlapping parts of ontologies by finding equivalents entities (concepts, attributes and relations)

Ontology mediation: The whole process of designing ontology mappings plus using them at run-time.

Data mediation: Synonym of ontology mediation in a web services context

Application Programming Interface (API): Program organized in libraries designed to be reusable for tool building.

Extended Backaus-Naurus Form (EBNF): Normal form to represent grammars representing machine languages.

WSMO: Web Services Modelling Ontology, the semantic web services framework developed in the DIP project.

WSMO4j: The ontology model implementing the WSMO framework

OMWG: The Ontology Management Working Group is an initiative leaded by DERI Innsbruck to extend the Ontology Management WP2 of the Dip Project.

SEKT: Semantically Enabled Knowledge Technologies, European Project part of the ESSI cluster

ESSI: European Semantic Systems Initiative is grouping different EU funded projects: DIP, SEKT, Knowledge Web and ASG (Adaptive Services Grid).
# Table of Contents

**Executive Summary** ........................................................................................................ 1

**List of Key Words/Abbreviations** .................................................................................. 7

**Table of Contents** .......................................................................................................... 8

**1 Introduction** .................................................................................................................. 9

**2 General Requirements** ................................................................................................ 9

2.1 Interoperability and Compatibility ................................................................................. 9

2.2 Generivity ......................................................................................................................... 10

3. Functional Requirements ............................................................................................... 10

3.1 Ontology Mapping Language Requirements ............................................................... 10

3.2 Graphical Mapping Interface ......................................................................................... 12

3.3 Mapping Discovery ......................................................................................................... 12

4. Global Architecture ......................................................................................................... 14

5. Ontology Mapping specification Language .................................................................... 15

5.1 Simple Mappings .......................................................................................................... 15

5.2 More Complex Mappings .............................................................................................. 18

5.3 Instances Transformation Functions ............................................................................ 19

5.4 RDF Syntax .................................................................................................................... 20

6. Implementation ................................................................................................................ 24

6.1 Mapping API ................................................................................................................ 24

6.2 Graphical Mapping Tool ............................................................................................. 26

7. Conclusion and Perspectives .......................................................................................... 26

References .......................................................................................................................... 27

Annex1 Mapping Language EBNF Grammar .................................................................... 29

## List of Figures

Global mapping architecture .............................................................................................. 6

Mapping Architecture ....................................................................................................... 18
1. Introduction

Deliverable 2.6 includes tools and documents related to ontology mapping part of the DIP ontology management system. This document gives a theoretical grounding to 2.6 deliverable. The first version of this document was introducing the basic constructs for ontology mapping specification together with a set of requirements. In this new version we extend these requirements to cope with specific mappings for automatic mapping discovery and web services specificities. We present the mapping language revised to take them into account.

Section 2 gives requirements for a mapping tool in terms of functionality, compatibility and genericity. Section 3 give some more particular requirements, about the mapping language, the graphical interface and the need to automatize mappings. Section 4 gives an overview of the architecture and section 5 presents the mapping language. Section 6 gives some insights on the implementation. Section 7 concludes this document.

2. General Requirements

This section presents general requirements in terms of interoperability, compatibility and genericity.

2.1. Interoperability and compatibility

The Ontology management suite developed in the context of dip\(^1\) is composed of different parts that form a complete ontology management system (OMS): ontology editing and browsing, ontology mappings and ontology versionning. The ontology mapping module interacts with the different parts of the ontology management system, giving some first requirements to its design. We give at the end of each requirement a pointer to the related work in the DIP project. The reader may refer to these documents for a better understanding of the stated requirements.

The ontology mapping module communicates with the ontology editor. The mappings that are realized between ontologies are implying having a representation of the entities (concepts, attributes, instances) to be related. The ontology editor gives such a representation that can be reused by the mapping editor. We may there envisage two possible realization. In the first case, the mappings functionality may be an embedded feature of the ontology editor. In the second case, mapping editor is a separate component that invokes some features of the ontology editor in order to display the concept tree of the ontologies to be related. For more informations about the ontology editor, please have a look to [11]

The ontology mapping module can load ontologies from multiple sources. In order to realize the mappings, the user should first be able to open the ontologies to be mapped. Ontologies being stored as files in a classical OS can be retrieving by loading the file from the file-system. There should also be the possibility to open an ontology at a given URL. Ontologies in the Dip OMS are stored in an ontology repository (see [6]). The ontology mapping module should then be able to communicate with the ontology repository, to retrieve the stored ontologies that will be used to create the mappings. The same happen when the user open a previously stored mapping (see next point), the mapping module should automatically retrieve the ontologies associated with this module and open them using the editor. For more information, please have a look to the ontology repository [6].

\(^1\)See the Ontology Management Working Group: http://www.omwg.org
The ontology mapping module communicates with the ontology mapping store. Once the mappings created, they need to be stored. An ontology mapping store should then be introduced as part of the OMS, giving functionalities to store the created mapping and retrieve them in a convenient automatic way. When retrieving these stored mappings the user may want to visualize or modify them. For more informations about the mapping store, please have a look to [12].

The ontology mapping module communicates with the ontology versioning support module. Ontologies are dynamic objects that evolve over time as the domain they conceptualize evolves. The versioning module is taking care of keeping the trace of this evolution. On a conceptual level, it may be considered that two domain-related ontologies one one hand, and an ontology on which the user has performed small changes on the other hand are very similar. In other words, differences between 2 versions of the same ontology can be represented as mappings between these 2 versions. For more informations about the ontology versioning support, please have a look to [3].

2.2. Genericity

We identify two aspects on which the ontology mapping tool must be open.

Mapping representation

An important feature of the DIP mediation component will be the reusability of the mappings generated by the users in other tools (reasoner, semantic web services builder, etc). In that purpose, we add as a requirement the commitment to an abstract mapping representation language, independent with the ontology language. Such an abstract language will give us the possibility to represent complex mapping expression in an abstracted way, independently from the ontology language. These mappings will of course have to be grounded at the run-time to the language of the ontologies they are dealing with.

Pluggable mapping Architecture

The research in the mediation field is mainly concentrated on the automation of the mapping task. This research is active and involve different areas like graph theory or computational linguistics. As a result new algorithm realizing automatic or semi-automatic ontology mappings are continuously proposed. To get the advantage of these new algorithms, the DIP mapping tool needs an architecture that permits to plug the automation algorithms and select an appropriate one for a specific mapping task.

3. Functional Requirements

In this section we elaborate some the requirements of the DIP mapping tool related with the functionalities we will offer. As specified in the last requirement, we impose the use of an abstract mapping language. From this language and a study of the mappings that are recurrently created by the user, we propose a library of mapping patterns. This library helps the user to select the appropriate mapping and then considerably reduce the time spent on the mapping task. The time spent on the mapping task will also be reduced by using automatic or semi-automatic ontology mapping algorithms, we also consider to present to the user a graphical mapping interface with the schema of the ontologies to be mapped. We will now describe these functionalities in more details.

3.1. Ontology Mapping Language Requirements

The core of the mapping tool is the abstract mapping language. It gives a format to represent the mappings at an abstract level, independent from the ontology language
used. We will in the following try to give a list of requirement that such a language must fulfil to be efficient.

**Mapping on the semantic web** Our goal is to develop a mapping language for the semantic web. As different ontologies representation language are in use in the semantic web\(^2\), the semantic web mapping language must give the possibility to realize mapping between ontologies written in these languages. For this we propose the use of an abstract mapping language ontology-language independent that may be grounded to different ontology languages depending on the needs of the user.

**Specify instances transformations** The mapping process is specified in [9] as the set of activities required to transform instances of the source ontology into instances of the target ontology. The mapping language must then give the necessary constructs to express this transformation. In instance transformation, we identify two dimensions: structural transformation and value transformation. A *structural* transformation is a change in the structure of an instance. This means that an instance might be split into two instances, two instances must be merged into one, properties might be transformed to instances, etc. For example an instance of the concept PdD-Student in one ontology might need to be split into two instances, one of Student and one of Researcher, in the target ontology. Another example is the use of aggregate functions. An ontology Os might have a concept Parent with a property hasChild, whereas the ontology Ot might also have a class Parent, but in this case only with the property nrOfChildren. An aggregate function is required to count the number of children in Os in order to come with a suitable property filler for nrOfChildren. A *value* transformation is a simple transformation from one value into another. This transformation affects attribute values of the instances. An example of such a value transformation is the transformation from kilograms into pounds. An example of a transformation, which requires both a structural and two value transformations is the transformation from a full name to separate first and last names. Splitting the full name into both the first and the last names requires structural transformation. After the structural transformation, two value transformation are required; one for the first and one for the last name.

Specifying instance transformation doesn't avoid to map the concepts, attributes or relation categorizing the instances. It is rather an additional functionality allowing the user to specify the mapping better in order to get correct executable mappings.

**Query Rewriting and ontology merging** A query addressed to an ontology Os has to be rewritten in terms of the ontology Ot in order to get results both from Os and Ot. This use case indicate the need for the ontology mapping to not only map instances of the ontology but also map concepts and relations in the source and target ontologies. This is necessary for the case when a query written in terms of an ontology Os must be executed on an instance base, which is described by Ot. The mapping needs to specify exactly how concepts and relations in Os relate to concepts and relations in Ot in order to enable the rewriting. The mapping language is then required to provide the possibility of expressing the correspondences between the concepts and relations of the two ontologies.

**Mappings to version Ontologies** The ontology mapping language must support constructs to maintain compatibility between two versions of an ontology. We can illustrate this by giving the example of a concept dropped from an ontology. The new ontology version without this concept may be related to the old version by mapping the

---

\(^2\)the W3C standard Ontology Web Language (OWL) [8] or the Web Rule Language (WRL [2]) for example
dropped concept to its superconcept in the new version. This mapping is realized using a mapping language construct. As this task is concerning the versioning module of DIP we refer you to [3] for more details about this process.

**Treating classes as instances** Different ontologies might be modeled within slightly different domains with different granularity. What is seen as a class in one ontology might be seen as an instance of another class in another ontology. In order to support inter-operation between two ontologies with such differences, class need to be mapped to instances and vice-versa. In a more general way, the mapping language should support mapping of any entity in the source ontology to any entity in the target ontology. For example it should be possible to have a relation instance mapping, a class-relation mapping, etc.

**Mappings of different cardinalities** It might be necessary to map a class in one ontology to a number of different classes in the other ontology. It might also be necessary to map a class in one ontology to a class and a relation in the other ontology. In other words, the language needs to support mappings of arbitrary cardinalities.

**Conditional mappings** In some cases the mapping relation between an entity of the source ontology and another entity of the target ontology may apply only if certain conditions are satisfied. For example the concept human in the source ontology is mapped to the concept man in the target ontology only if the attribute gender of the concept human is equal to male. This conditional mapping should be expressed in the mapping language.

**Mapping Patterns** Patterns are templates that match the more usual mismatches between two ontologies. The use of predefined patterns is expected to considerably reduce the mapping designer task. These patterns should be expressed in the mapping language, they may be seen as building blocks for this language. A pattern is instantiated for a particular pair of ontologies into an ontology mapping, a complex mapping pattern is composed of other (elementary or complex mapping patterns) and a mapping is composed of a number of instantiated mapping patterns. The ontology mapping language must supports the use of elementary patterns as constructs of the language itself. A pattern specifies the relation between one or more ontology constructs of the source ontology and one or more ontology constructs of the target ontology. The mapping pattern describes both the relation and the necessary instance transformation in a declarative way. When the mapping pattern is instantiated to a mapping relation, the abstract ontology constructs in the patterns (empty ontological references) are filled-in with the actual ontology constructs from the source and target ontologies. An additional benefit that these mapping patterns could bring is when mapping patterns would be associated with similarity detection methods. A Match operator could then detect mapping patterns based on certain similarities between the different ontologies.

### 3.2. Graphical Mapping Interface

Mapping between two ontologies consists in relating the corresponding entities from one ontology to the other. The abstract mapping language gives a format to express the complex relation that may stand between these entities. The user of the DIP mapping tool should have the possibility of creating mappings by directly entering constructs of the language or by using patterns in a graphical manner. For ergonomic the DIP mapping tool should also include a graphical interface for this task. A ontology mapping graphical interface is commonly constituted of two representations of the ontology to be mapped (see for example [7]). For more informations about the used representation of the ontology, see the ontology editing module of DIP [11]. The user
select the correspondent entities in the two ontologies to realize the mapping. When the
correlation standing between the two ontologies is complex, the user may select a
convenient pattern and apply it to the mapping case.

3.3. Mapping Discovery

The large ontologies like Proton [14] or Wordnet [5] contain thousands of concepts and
relations. It is obviously a complex and time consuming task to map two ontologies
having these sizes, almost each concepts and relations in the source ontology having a
correspondent that must be determined in the target ontology. In that scope, the
assistance of automated method to (semi-)automatically find the correspondences
between the entities of two ontologies would be very welcome. Automatically means
using algorithms that discover correspondences between the ontological entities without
human intervention. Semi-automatically means that the user have to validate and
complete the mappings proposed by an algorithm. As already mentioned, the research
on these methods is very dynamic and different algorithms are trying cope with this
problem.

There are different techniques used to automatically find mappings between ontology
schema. A good classification of these techniques is given in [13]. The techniques are
classified in two categories: Element-level and Structure-Level. The former technique
use the labels and descriptions of the ontological entities and compare them, eventually
using some external tools to measure the distances between two elements from two
ontologies. An example of a method element-level based method would be to compare
the labels of the two ontologies pair by pair using a string comparison method. The
latter technique consists in analysing the structure of the ontological schema, by
matching parts of the concept tree for example. These techniques are then returning
similarity measures for the entities of two ontologies, these measures are then propose
to the user which validates the effective mapping.

DIP mediation module will have an interface on which such automation algorithms
be plugged. This interface takes one or more algorithms as an entry together with
the indication about which technique of the classification the algorithm is actually
using. The algorithms may then be combined to cover the different techniques and
compute better quality mappings, ideally fully automatized.

4. Global architecture

Here is the architecture diagram of the DIP mapping tool, including the required
components presented in this document.

• The graphical mapping interface helps the user in creating mappings
• The mappings are stored in the mappings and patterns repository
• The ontologies are stored in the ontology repository
• The mapping API deals with mapping documents (parsing/serializing) and
discover new mappings using the algorithms plugged-in.

The implementation delivered with this document uses this architecture.
We present in this section the language developed on the basis of the requirements elicited in the first part of this document. The language specification is given together with explanation about the role of the different constructs, in section 5.2 we give the core specification of the language that allow to express simple entity to entity mappings. In section 5.3 we extend this specification, allowing to map between many to many entities between the source and target ontologies. A set of operator is introduced to link together those entities. Section 5.4 introduce constructs to condition the validity of the mappings and section 5.5 introduce the specification of instance transformations as a set of services. A full specification of the language in EBNF style is available in annex. For tools and examples of this language, please consult the section 6.

5.1. Simple mappings

We present here the mapping language developed to express correspondences between ontologies. This language is being developed as a joint effort in the WSMO and OMWG working groups, and the SEKT and DIP European projects.

We present the abstract syntax of the language written in EBNF, similar to the WSML (http://www.wsmo.org/wsml/wsml-syntax) abstract syntax. Any element between square brackets ‘[’ and ‘]’ is optional. Any element between curly brackets ‘{’ and ‘}’ can have multiple occurrences. Each element of an ontology on the Semantic Web, whether it is a class, attribute, instance, or relation, is identified using a IRI which is an extension of the URI format. In the abstract syntax, a IRI is denoted with the name IRIReference. We define the following identifiers:

```
documentid = iri
```
We allow concrete data values. The abstract syntax for data values is taken from the OWL abstract syntax:

```
Listing 2. Literals

{typedliteral} typedliteral
| {plainliteral} plainliteral

literal = | {numeric} number
| {string} string

typedliteral = plainliteral \iri
plainliteral = " literal_content* " language_tag?
```

The lexical form is a sequence of Unicode characters in normal form C, as in RDF. See http://www.w3.org/TR/charmod/ for more information. The language tag is an XML language tag, as in RDF.

First of all, the mapping document itself is declared, along with the ontologies participating.

```
Listing 3. Mapping Document

mappingdocument = t_mappingdocument ( documentid source_exp target_exp
annotation* expression* )

source_exp = source ( ontologyid )

target_exp = target ( ontologyid )
```
A mapping consists of a number of annotations, corresponding to non-functional properties in WSMO (http://www.wsmo.org/), and a number of mapping rules. The creator of the mapping is advised to include a version identifier in the non-functional properties as well as other indications about the nature of the mappings.

**Listing 4. Annotation**

```plaintext```
annotation = t_annotation ( propertyid propertyvalue )
```

The Alignment Format is a format to represent ontology mappings proposed in [4] and accessible at http://co4.inrialpes.fr/align/format.html. It is designed to express mappings resulting from matching algorithms. To comply with this format we introduce a special annotation. The **measure** gives an indication about the level of confidence of a mapping rule, it is given as a real number in [0,1]. This field is mainly used when the mappings are the results of a matching algorithm. It may also be proposed to a user in a graphical tool, if the mapping is complex and the user is not certain about his current modelling. The alignment format contains other field that compatible with the mapping language format. The **relation** gives the kind of relation standing between the two mapped expression. This information is given in the mapping language by the directionality field (see below).

**Listing 5. Measure**

```plaintext```
measure = t_measure ( float )
```

A mapping can be either uni- or bidirectional. A unidirectional mapping from a source to a target means that the source expression is subsumed by the target one. This corresponds to the inclusion in the alignment format relation field. A bidirectional mapping means the source and the target expressions are equivalent. This corresponds to the equality in the alignment format relation field.

**Listing 6. Directionality**

```plaintext```
directionality = {unidirectional} unidirectional | {bidirectional} bidirectional
```

Expressions are either class mappings, relation mappings, instance mappings or arbitrary logical expressions. The syntax for the logical expressions is not specified; it depends on the actual logical language to which the language is grounded. A special kind of relation mappings are attribute mappings. Attributes are binary relations with a defined domain and are thus associated with a particular class. In the mapping itself the attribute can be either associated with the domain defined in the (source or target) ontology or with a subclass of this domain. In order to distinguish these kinds of mappings, we introduce two different keywords for class, relation and attribute mappings, namely ‘unidirectional’ and ‘bidirectional’. Individual mappings are always bidirectional.

---

1. We consider in this document that Ontology Alignment and Ontology Mapping refer to the same concept
It is possible, although not required, to nest attribute mappings inside class mappings. Furthermore, it is possible to write an axiom, in the form of a class condition, which defines general conditions over the mapping, possibly involving terms of both source and target ontologies. Notice that this class condition is a general precondition for the mapping and thus is applied in both directions if the class mapping is a bidirectional mapping. Notice that we allow arbitrary axioms in the form of a logical expression. The form of such a logical expression depends on the logical language being used for the mappings and is thus not further specified here.

[first] and [second] are introduced here to help the parser disambiguating between the two expression.

Listing 7. Expressions

\[
\text{expression} = \begin{cases}
\{\text{logical_expression}\} & \{\text{annotation} \} \{\text{logical_expression} \} \\
\text{classmapping} ( \{\text{annotation}\} \{\text{measure}\}\{\text{directionality}\}\{\text{first}\}: \text{classexpr} \{\text{second}\}: \text{classexpr}\{\text{logicalexpressionbrace}\}? ) \\
\text{attribute_mapping} ( \{\text{annotation}\} \{\text{measure}\}\{\text{directionality}\}\{\text{first}\}: \text{attributeexpr} \{\text{second}\}: \text{attributeexpr}\{\text{attributecondition}\}\{\text{transformation}\}\{\text{logicalexpressionbrace}\}? ) \\
\text{relation_mapping} ( \{\text{annotation}\} \{\text{measure}\}\{\text{directionality}\}\{\text{first}\}: \text{relationexpr} \{\text{second}\}: \text{relationexpr}\{\text{relationcondition}\}\{\text{logicalexpressionbrace}\}? ) \\
\text{classattribute_mapping} ( \{\text{annotation}\} \{\text{measure}\}\{\text{directionality}\}\{\text{first}\}: \text{attributeexpr} \{\text{second}\}: \text{classexpr}\{\text{logicalexpressionbrace}\}? ) \\
\text{classrelation_mapping} ( \{\text{annotation}\} \{\text{measure}\}\{\text{directionality}\}\{\text{first}\}: \text{relationexpr} \{\text{second}\}: \text{classexpr}\{\text{logicalexpressionbrace}\}? ) \\
\text{classinstance_mapping} ( \{\text{annotation}\} \{\text{measure}\}\{\text{directionality}\}\{\text{first}\}: \text{instanceid} \{\text{second}\}: \text{instanceid}\{\text{logicalexpressionbrace}\}? ) \\
\text{attributeclass_mapping} ( \{\text{annotation}\} \{\text{measure}\}\{\text{directionality}\}\{\text{first}\}: \text{attributeexpr} \{\text{second}\}: \text{classexpr}\{\text{logicalexpressionbrace}\}? ) \\
\text{relationclass_mapping} ( \{\text{annotation}\} \{\text{measure}\}\{\text{directionality}\}\{\text{first}\}: \text{relationexpr} \{\text{second}\}: \text{classexpr}\{\text{logicalexpressionbrace}\}? ) \\
\text{instanceclass_mapping} ( \{\text{annotation}\} \{\text{measure}\}\{\text{directionality}\}\{\text{first}\}: \text{instanceid} \{\text{second}\}: \text{classexpr}\{\text{logicalexpressionbrace}\}? ) 
\end{cases}
\]
There is a distinction between attributes mapping in the context of a class and attributes mapped outside the context of a particular class. Because attributes are defined locally for a specific class, we expect the attribute mappings to occur mostly inside class mappings. The keywords for the mappings are the same. However, attribute mappings outside of the context of a class mappings need to be preceded with the class identifier, followed by a dot "."

Simple mapping expressions are expressed by replacing the class, attribute and relation expressions with corresponding identifiers of the respective class attribute or relation. We see in the next section how to expand these simple mappings, going deeper in the definition of the class, relation and attribute expressions.

### 5.2 More complex mappings

We here introduce more constructs to express complex mappings. These constructs allow us to express mappings at the level 1 (simple entity to entity mapping) and 2 (set of entities to set of entities mapping) as defined in [4].

For class expressions we allow basic boolean algebra. This corresponds loosely with Wiederhold’s ontology algebra [15]. Wiederhold included the basic intersection and union, which correspond with our and and or operators. Wiederhold’s difference operator corresponds with a conjunction of two class expressions, where one is negated, i.e. for two class expressions C and D, the different C-D corresponds with \( \text{and}(C,\neg(D)) \).

The join expression is a specific kind of disjunction. A join expression includes a logical expression which specify which instances are included in the disjunction when the mapping is executed.

#### Listing 8. Class expression

```plaintext
classexpr = {classid} classid |
            {and} and ([first]: classexpr [second]: classexpr classexpr*) |
            {or} or ([first]: classexpr [second]: classexpr classexpr*) |
            {not} not (classexpr)
```

Attribute expressions are defined as such, allowing for inverse, transitive close, symmetric closure and reflexive closure, where inverse(A) stands for the inverse of A, symmetric(A) stands for the symmetric closure of A, reflexive(A) stands for the reflexive closure of A and trans(A) stands for the transitive closure of A. Especially when mapping several source ontologies into one target ontology, different classes and relations need to be joined. Although apparently similar, a join mapping is from an intersection mapping, that the first entity expression will be intersected the other entities.

#### Listing 9. Attribute expression

```plaintext
attributeexpr = {attributeid attributeid} |
                {and} and ([first]: attributeexpr [second]: attributeexpr)
```

1. 18
Relation expressions are defined similar to class expressions, with the difference that the arity of the relations may be precised.

Listing 10. Relation expression

```
{relationid}

| {and} and ([first]: relationexpr [second]: relationexpr) 

relationexpr = | {or} or ([first]: relationexpr [second]: relationexpr*)  

| {not} not (relationexpr) 

| {join} join ([first]: relationexpr [second]: relationexpr* logicalexprbrace? ) 

arity = [arity_val]
```

We now introduce the conditions that may be applied to the mappings. A mapping is valid if the condition is satisfied. Conditions may apply on attributes in class or attribute mappings.

Listing 11. Conditions

```
classcondition = attributevaluecondition (attributeid indidordatalittorclassexpr )  

attributecondition = expressioncondition (attributeexpr )
```
5.3. Instance Transformation Functions

As presented in the requirements there is a need to specify transformations of instances in the mappings definition. We will in this section give a set of transformation functions explicating the instance transformations between the two ontologies to be mediated. We propose two way of specifying the transformations: using a library of transformations functions or using a web service. In a first approach, we will propose an extensible library of functions to be included in the pattern library for storage and retrieval. These functions should cover most of the cases presented in the hierarchy of transformation functions in the document [12]. the only aspect not covered by this library is the dynamical aspect some transformations require (evolution of currency change rate for example, time related values in general). We cope with this aspect in the second way we propose to handle instance transformation, namely by specifying a web service.

Listing 13. Instance transformation functions

\[
\text{transformation} = \\
\begin{cases}
  \{\text{function}\} & \text{t_transformation (functionid param*)} \\
  \{\text{service}\} & \text{t_transformation ( service iri param*)}
\end{cases}
\]

\[
\begin{align*}
\text{functionid} &= \{\text{string}\} \text{ string} \\
&\quad | \{\text{iri}\} \text{ iri} \\
\text{param} &= \{\text{string}\} \text{ string} \\
&\quad | \{\text{iri}\} \text{ iri} \\
&\quad | \{\text{number}\} \text{ number}
\end{align*}
\]

The set of functions is to be implemented. The web service protocol is to be chosen at the implementation level, we however recommend to use a semantic based web services framework like the Web Service Modeling Ontology, WSMO [WSMO2005]

5.4. RDF Syntax

We give in this section the RDF translation of the mapping language abstract syntax. The implementation of the mapping language will support import/export from this format as well.

RDF is the basic metadata interchange format on the semantic web. It is at a lower level than the ontology language (WSML in our case). We give a RDF definition of the mapping language to show its compatibility with this standard, and towards an RDF/XML implementation. Having an RDF syntax will favorize the spreading of the language.

Here is the list of keywords. We use 'map' as a prefix. The URI corresponding to this prefix is the URI of the current document: http://www.omwg.org/TR/d7/d7.2.
### Keyword | Comment
--- | ---
map#mappingDocument | used to define a mapping document
map#classMapping | mapping between two class expressions
map#attributeMapping | Mapping between two attribute expressions
map#relationMapping | mapping between two relation expressions
map#individualMapping | mapping between two instances
map#classAttributeMapping | mapping between a class expression and an attribute expression
map#classRelationMapping | mapping between a class expression and a relation expression
map#classInstanceMapping | mapping between a class expression and an instance
map#onto1 | first ontology
map#onto2 | second ontology
map#directionality | indicates the directionality of a mapping rule
map#measure | indicates the confidence given to a mapping rule
map#operator | indicates the operator in a complex class, attribute or relation expression
map#condition | specify a condition
map#hasSource | source expression of a mapping rule
map#hasTarget | target expression of a mapping rule
map#hasExpression | source or target expression of a mapping rule. Can also be a sub-expression of an expression
map#logicalExpression | kind of expression, represented as a string

The remainder of this section present the translation from the abstract syntax to the rdf syntax. We use the existing rdf constructs to limit the number of introduced keywords. In particular, the documentid is mapped to dc#title, the annotation is mapped to rdf#description.

A, B, C represent identifiers, $DV_i$ stands for an integer value, $DV_d$ stands for a decimal, and $DV_s$ stands for a string data value, and $n$ is an integer number.

<table>
<thead>
<tr>
<th>Abstract Syntax</th>
<th>RDF Triples</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>T( MappingDocument(</td>
<td>$A$ rdf#type $A$ is the IRI representing the map#mappingDocument mapping document given as an id.</td>
<td></td>
</tr>
</tbody>
</table>
Annotations are used to indicate informations about the document and the rule, the propertyid $B$ is a dublin core property and then may be used here as an RDF predicate.
The blank identifier :X denotes a helper node to bind the mapping rule to the mapping document.
6. Implementation

This section briefly presents the tools developed to manage ontology mappings.

6.1. Mapping API

The mapping language API is the tool allowing to deal with the mapping language. It provides an in-memory representation of the language constructs as well as parsing and serializing functionalities.

Here is the architecture picture of the mapping language API\(^4\). Wrappers give the possibility to construct mappings using objects from a particular ontology language API. In the framework of DIP we use WSMO4j\(^5\) through the WSML wrapper. Other wrappers can easily be implemented on top of this API to support other ontology languages, like OWL for example. The exporter allows to generated mapping expressed

\(^4\)This API is available at jar file at the following address: http://dome.sourceforge.net/snapshot/

\(^5\)WSMO4j :http://wsmo4j.sourceforge.net

\[T(\text{attributeid}, A)\]

\[A \text{ map#hasExpression attributeid} \]

\[A \text{ map#operator map#and} \]

\[A \text{ map#hasExpression} \]

\[T(\text{and}(\text{attributeExpr}_{i} \ldots \text{attributeExpr}_{n}, A)\]

\[\text{Identical for the other operators, just change to map#_operator where operator is the corresponding operator. See the attribute operators list.} \]
in different formats in order to be executed in a reasoner. At this day mapping can be expressed in the mapping language, WSML and OWL. The core of the API is the Object Model which present an in-memory representation of the mappings and allow to manipulate them as Java objects. Details for downloading the API are given in the Fact Sheet coming with the deliverable 2.6.

Illustration 2: Mapping API Architecture
6.2. Graphical Mapping Tool

Developed in DIP, the graphical mapping tool uses the mapping language API to construct mappings. The purpose of this tool is to edit mapping documents using either a table view representing the document, or a double tree view on which the two ontologies to be related are displayed. The user realizes mapping by selecting the appropriate concepts. Installation and usage details of the tool are provided online at the following address:

http://www.unicorn.com/dip/eclipse_plugins/v0.2/20060101/MappingEditorFactSheet.html

7. Conclusion and Perspectives

This third version of the mapping theory document has presented refined requirements for correctly expressing the mappings at the basis of the tools to be developed as part of 2.6 deliverable. We have added complex mappings specification to the mapping language as well as special fields giving the possibility to express instance transformation for web services mediation. At the time this document is written the implementation of the tools is on its final phase. The mapping language API reached a stable version, while the graphical user interface is on testing process, implementing the ideas formulated along the different versions of this theory document.

\[^6\) This tool is described in a separate document available at

http://www.unicorn.com/dip/eclipse_plugins/v0.2/20060101/MappingEditorFactSheet.html
REFERENCES


### Annex 1: Mapping Language EBNF Grammar

#### Helpers
- **all** = [0x0 .. 0xffff]
- **escape_char** = '
-
- **basechar** = [0x0041 .. 0x005A] | [0x0061 .. 0x007A]
- **ideographic** = [0x4E00 .. 0x9FA5] | 0x3007 | [0x3021 .. 0x3029]
- **letter** = basechar | ideographic
- **digit** = [0x0030 .. 0x0039]
- **combiningchar** = [0x0300 .. 0x0345] | [0x0360 .. 0x0361] | [0x0483 .. 0x0486]
- **extender** = 0x00B7 | 0x02D0 | 0x02D1 | 0x0387 | 0x0640 | 0x0E46 | 0x0EC6 | 0x3005 | [0x3031 .. 0x3035] | [0x309D .. 0x309E] | [0x30FC .. 0x30FE]
- **alphanum** = digit | letter
- **hexdigit** = [0' .. '9'] | ['A' .. 'F']
- **not_escaped_ncnamechar** = letter | digit | '_' | combiningchar | extender
- **escaped_ncnamechar** = '.' | '-' | not_escaped_ncnamechar
- **ncnamechar** = (escape_char escaped ncnamechar) | not_escaped_ncnamechar
- **reserved** = '?' | '?' | '#' | '[' | ']' | ':' | '@' | '&=' | '=' | '+' | '$' | ','
- **mark** = '_' | '_' | '.' | '.'
- **escaped** = '%' hexdigit hexdigit
- **unreserved** = letter | digit | mark
- **scheme** = letter (letter | digit | '+' | '-' | '.')*
- **port** = digit*
- **idomainlabel** = alphanum ((alphanum | '-')*)? alphanum
- **dec_octet** = digit | ([0x31 .. 0x39] digit | ('1' digit digit) | ('2' [0x30 .. 0x34] digit) | ('25' [0x30 .. 0x35])
- **ipv4address** = dec_octet '.' dec_octet '.' dec_octet '.' dec_octet
- **h4** = hexdigit hexdigit? hexdigit? hexdigit?
- **ls32** = (h4 ':' h4) | ipv4address
dquote = ""'
not_cr_lf = [ all - [ cr lf ] ]
escaped_char = escape_char all
not_escape_char_not_dquote = [ all - [ """ + escape_char ] ]
not_escape_char_not_squote = [ all - [ "" + escape_char ] ]
literal_content = escaped_char | not_escape_char_not_dquote
string_content = escaped_char | not_escape_char_not_squote
not_star = [ all - '*' ]
not_star_slash = [ not_star - '/' ]
long_comment = '/\* not_star*\*\*' + ( not_star_slash not_star*\*\*' + )* '/\*
begin_comment = '/\* | 'comment'
short_comment = begin_comment not_cr_lf* eol
comment = short_comment | long_comment
blank = ( ' ' | tab | eol )+
qmark = '?'
luridel = '<''
ruridel = '>''
primary_subtag = letter+
language_subtag = ( letter | digit )+
language_tag = '@' primary_subtag ( '.' language_subtag )*

• Tokens
t_blank = blank
t_comment = comment
comma = ','
endpoint = '. ' blank
pathcon = '.'
dblcaret = '^\^'
lpar = '('
rrpar = ')'
relationclassmapping = 'relationClassMapping'
instanceclassmapping = 'instanceClassMapping'
t transformation = 'transformation'
service = 'service'
and = 'and'
or = 'or'
not = 'not'
join = 'join'
inverse = 'inverse'
symmetric = 'symetric'
transitive = 'trans'
reflexive = 'reflexive'
univ_false = 'false'
univ_true = 'true'
attributevaluecondition = 'attributeValueCondition'
attributeoccurencecondition = 'attributeOccurrenceCondition'
valuecondition = 'valueCondition'
expressioncondition = 'expressionCondition'
string = squote string_content* squote
plainliteral = dquote literal_content* dquote language_tag?
full_iri = luridel iri_reference luridel
ncname = ( letter | '_' ) ncnamechar*
anonymous = '_#' digit*
pos_int = digit+
pos_float = digit+ '.' digit+

* Ignored Tokens
  • t blank
  • t comment

* Productions
mappingdocument = t_mappingdocument lpar documentid source_exp target_exp annotation* expression* rpar
source_exp = source lpar ontologyid rpar
**Mapping Tool Background**

\[
\text{target}\_\text{exp} = \text{target}\ lpar\ \text{ontology}\ id\ rpar
\]

\[
\text{annotation} = t\_\text{annotation}\ lpar\ \text{property}\ id\ \text{propertyvalue}\ rpar
\]

\[
\text{measure} = t\_\text{measure}\ lpar\ \text{float}\ rpar
\]

\[
\{\text{logical}\_\text{expression}\} = \{\text{annotation}\*\ \text{logical}\_\text{expression}\ \text{rbrace}\}
\]

\[
\{\text{class}\_\text{mapping}\} = \text{classmapping}\ lpar\ \text{annotation}\*\ \text{measure}\ ?\ \text{directionality}\ ?\ [\text{first}]:\ \text{classexpr}\ [\text{second}]:\ \text{classexpr}\ \text{classexpr}\ *
\]

\[
\{\text{attribute}\_\text{mapping}\} = \text{attributemapping}\ lpar\ \text{annotation}\*\ \text{measure}\ ?\ \text{directionality}\ ?\ [\text{first}]:\ \text{attributeexpr}\ [\text{second}]:\ \text{attributeexpr}\ \text{attributecondition}\*\ \text{transformation}\ ?\ \text{logical}\_\text{expression}\ \text{rbrace}\ ?\ rpar
\]

\[
\{\text{relation}\_\text{mapping}\} = \text{relationmapping}\ lpar\ \text{annotation}\*\ \text{measure}\ ?\ \text{directionality}\ ?\ [\text{first}]:\ \text{relationexpr}\ [\text{second}]:\ \text{relationexpr}\ \text{relationcondition}\*\ \text{logical}\_\text{expression}\ \text{rbrace}\ ?\ rpar
\]

\[
\{\text{instance}\_\text{mapping}\} = \text{instance}\_\text{mapping}\ lpar\ \text{annotation}\*\ [\text{first}]:\ \text{instance}\ id\ [\text{second}]:\ \text{instance}\ id\ \text{rpar}
\]

\[
\{\text{classattribute}\_\text{mapping}\} = \text{classattributemapping}\ lpar\ \text{annotation}\*\ \text{measure}\ ?\ \text{directionality}\ ?\ [\text{first}]:\ \text{classexpr}\ [\text{second}]:\ \text{attribute}\_\text{expr}\ \text{logical}\_\text{expression}\ \text{rbrace}\ ?\ rpar
\]

\[
\{\text{classrelation}\_\text{mapping}\} = \text{classrelationmapping}\ lpar\ \text{annotation}\*\ \text{measure}\ ?\ \text{directionality}\ ?\ [\text{first}]:\ \text{classexpr}\ [\text{second}]:\ \text{relation}\_\text{expr}\ \text{logical}\_\text{expression}\ \text{rbrace}\ ?\ rpar
\]

\[
\{\text{classinstance}\_\text{mapping}\} = \text{classinstancemapping}\ lpar\ \text{annotation}\*\ \text{measure}\ ?\ \text{directionality}\ ?\ [\text{first}]:\ \text{classexpr}\ [\text{second}]:\ \text{instance}\ id\ \text{logical}\_\text{expression}\ \text{rbrace}\ ?\ rpar
\]

\[
\{\text{attributeclass}\_\text{mapping}\} = \text{attributeclassexpression}\ lpar\ \text{property}\ id\*\ \text{measure}\ ?\ \text{directionality}\ ?\ [\text{first}]:\ \text{attribute}\_\text{expr}\ \text{logical}\_\text{expression}\ \text{rbrace}\ ?\ rpar
\]

\[
\{\text{relationclass}\_\text{mapping}\} = \text{relationclassexpression}\ lpar\ \text{property}\ id\*\ \text{measure}\ ?\ \text{directionality}\ ?\ [\text{first}]:\ \text{relation}\_\text{expr}\ \text{logical}\_\text{expression}\ \text{rbrace}\ ?\ rpar
\]

\[
\{\text{instanceclass}\_\text{mapping}\} = \text{instanceclassexpression}\ lpar\ \text{property}\ id\*\ \text{measure}\ ?\ \text{directionality}\ ?\ [\text{first}]:\ \text{instance}\ id\ [\text{second}]:\ \text{classexpr}\ \text{logical}\_\text{expression}\ \text{rbrace}\ ?\ rpar
\]

\[
\text{class}\_\text{expr} = \{\text{classid}\}\ \text{and}\ lpar [\text{first}]:\ \text{classexpr}\ [\text{second}]:\ \text{classexpr}\ \text{classexpr}\ *
\]

1.  33
\[
\begin{array}{l}
| \{ \text{or} \} | \text{or \ lpar \ [first]: \ classexpr \ [second]: \ classexpr \ classexpr* \ rpar} \\
| \{ \text{not} \} | \text{not \ lpar \ classexpr \ rpar} \\
| \{ \text{attribute} \} | \text{attributeid} \\
| \{ \text{and} \} | \text{and \ lpar \ [first]: \ attributeexpr \ [second]: \ attributeexpr \ attributeexpr* \ rpar} \\
| \{ \text{or} \} | \text{or \ lpar \ [first]: \ attributeexpr \ [second]: \ attributeexpr \ attributeexpr* \ rpar} \\
| \{ \text{not} \} | \text{not \ lpar \ attributeexpr \ rpar} \\
\text{attributeexpr} = | \{ \text{inverse} \} | \text{inverse \ lpar \ attributeexpr \ rpar} \\
| \{ \text{symetric} \} | \text{symetric \ lpar \ attributeexpr \ rpar} \\
| \{ \text{reflexive} \} | \text{reflexive \ lpar \ attributeexpr \ rpar} \\
| \{ \text{transitive} \} | \text{transitive \ lpar \ attributeexpr \ rpar} \\
| \{ \text{join} \} | \text{join \ lpar \ [first]: \ attributeexpr \ [second]: \ attributeexpr \ attributeexpr* \ logicalexprbrace? \ rpar} \\
| \{ \text{relationid} \} | \text{relationid \ arity?} \\
| \{ \text{and} \} | \text{and \ lpar \ [first]: \ relationexpr \ [second]: \ relationexpr \ relationexpr* \ rpar} \\
| \{ \text{or} \} | \text{or \ lpar \ [first]: \ relationexpr \ [second]: \ relationexpr \ relationexpr* \ rpar} \\
| \{ \text{not} \} | \text{not \ lpar \ relationexpr \ rpar} \\
| \{ \text{join} \} | \text{join \ lpar \ [first]: \ relationexpr \ [second]: \ relationexpr \ relationexpr* \ logicalexprbrace? \ rpar} \\
| \text{arity} | \text{= lbracket \ arity_val \ rbracket} \\
| \text{classcondition} | \text{= attributevaluecondition \ lpar \ attributeid \ indidordatalitterclassexpr \ rpar} \\
| \text{relationcondition} | \text{= string} \\
| \text{attributecondition} | \text{= expressioncondition \ lpar \ attributeexpr \ rpar} \\
| \text{transformation} | \text{= } \{ \text{function} \} \text{ t_transformation \ lpar \ functionid \ param* \ rpar} \\
| | \{ \text{service} \} \text{ t_transformation \ lpar \ service \ iri \ param* \ rpar} \\
| \text{iri} | \text{= full_iri} \\
| \text{id} | \text{= } \{ \text{iri} \} \text{ iri} \\
| | \{ \text{anonymous} \} \text{ anonymous} \\
| | \{ \text{literal} \} \text{ literal} \\
| | \{ \text{universal_truth} \} \text{ univ_true} \\
\end{array}
\]
<table>
<thead>
<tr>
<th>{universal_falsehood}</th>
<th>univ_false</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>ncname colon</td>
</tr>
<tr>
<td>lexicalform</td>
<td>plainliteral</td>
</tr>
<tr>
<td></td>
<td>{typedliteral} typedliteral</td>
</tr>
<tr>
<td>literal</td>
<td>{plainliteral} plainliteral</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>typedliteral</td>
<td>plainliteral dblcaret iri</td>
</tr>
<tr>
<td>neg_int</td>
<td>sub_op pos_int</td>
</tr>
<tr>
<td>float</td>
<td>sub_op? pos_float</td>
</tr>
<tr>
<td></td>
<td>{positive_int} pos_int</td>
</tr>
<tr>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>logicalexpression</td>
<td>string</td>
</tr>
<tr>
<td>logicalexprbrace</td>
<td>lbrace logicalexpression rbrace</td>
</tr>
<tr>
<td>directionality</td>
<td>{unidirectional} unidirectional</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>indidordatalitterclassexpr</td>
<td>iri</td>
</tr>
<tr>
<td>confidence</td>
<td>pos_float</td>
</tr>
<tr>
<td></td>
<td>{greaterthan} gt</td>
</tr>
<tr>
<td>relationaloperator</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>arity_val</td>
<td>pos_int</td>
</tr>
<tr>
<td>propertyvalue</td>
<td>string</td>
</tr>
<tr>
<td>documentid</td>
<td>iri</td>
</tr>
<tr>
<td>ontologyid</td>
<td>iri</td>
</tr>
<tr>
<td>classid</td>
<td>iri</td>
</tr>
<tr>
<td>propertyid</td>
<td>iri</td>
</tr>
<tr>
<td>attributeid</td>
<td>iri</td>
</tr>
<tr>
<td>relationid</td>
<td>iri</td>
</tr>
<tr>
<td>variable</td>
<td>type</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td>instanceid</td>
<td>iri</td>
</tr>
<tr>
<td>functionid</td>
<td>string</td>
</tr>
<tr>
<td>param</td>
<td>iri</td>
</tr>
</tbody>
</table>