Collaborative Project

LOD2 – Creating Knowledge out of Interlinked Data

Project Number: 257943  Start Date of Project: 01/09/2010  Duration: 48 months

Deliverable 4.4.1

Specification of the Mapping Publication and Discovery Framework

Project funded by the European Commission within the Seventh Framework Programme (2007 - 2013)
Abstract

The promise of the Web of Linked Data is to enable client applications to discover new data sources by following RDF links at run-time and to smoothly integrate data from these sources. Linked Data sources use different vocabularies to describe the same type of objects. It is also common practice to mix terms from different widely used vocabularies with proprietary terms. Thus Linked Data applications need to apply mappings to translate Web data to their local schema before doing any sophisticated data processing. Maintaining a local or central set of mappings that cover all Linked Data sources is likely to be impossible due to the size and dynamics of the Web of Linked Data. Thus we propagate a distributed, pay-as-you-go data integration approach where data publishers, vocabulary maintainers and third parties may publish expressive mappings on the Web. A client application which discovers data that is represented using terms that are unknown to the application may search the Web for mappings and apply the discovered mappings to translate data to its local schema. In order to realize this data integration paradigm, we specify in this design report a language for publishing expressive, named mappings on the Web and a composition method for chaining partial mappings from different sources based on a mapping quality assessment heuristic. In addition to the completely open Web use case, the mapping language can of course also be used to integrate data in more closed enterprise settings.
The information in this document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided “as is” without guarantee or warranty of any kind, express or implied, including but not limited to the fitness of the information for a particular purpose. The user thereof uses the information at his/ her sole risk and liability.

History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
<th>Revised by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>15/05/2011</td>
<td>Outline, initial references</td>
<td>Schultz, Isele</td>
</tr>
<tr>
<td>0.1</td>
<td>03/06/2011</td>
<td>Draft</td>
<td>Schultz, Isele</td>
</tr>
<tr>
<td>0.2</td>
<td>07/06/2011</td>
<td>Internal review</td>
<td>Bizer, Mendes</td>
</tr>
<tr>
<td>0.3</td>
<td>14/06/2011</td>
<td>Peer review</td>
<td>Peter Boncz (CWI)</td>
</tr>
<tr>
<td>0.4</td>
<td>25/06/2011</td>
<td>Address peer review comments</td>
<td>Bizer, Schultz, Isele</td>
</tr>
</tbody>
</table>

Author List

<table>
<thead>
<tr>
<th>Organisation</th>
<th>Name</th>
<th>Contact Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUB</td>
<td>Andreas Schultz</td>
<td><a href="mailto:a.schultz@fu-berlin.de">a.schultz@fu-berlin.de</a></td>
</tr>
<tr>
<td>FUB</td>
<td>Christian Bizer</td>
<td><a href="mailto:christian.bizer@fu-berlin.de">christian.bizer@fu-berlin.de</a></td>
</tr>
<tr>
<td>FUB</td>
<td>Robert Isele</td>
<td><a href="mailto:robertisele@googlemail.com">robertisele@googlemail.com</a></td>
</tr>
<tr>
<td>FUB</td>
<td>Pablo Mendes</td>
<td><a href="mailto:pablo.mendes@fu-berlin.de">pablo.mendes@fu-berlin.de</a></td>
</tr>
</tbody>
</table>
Executive Summary

The promise of the Web of Linked Data is to enable client applications to discover new data sources by following RDF links at run-time and to smoothly integrate data from these sources. Linked Data sources use different vocabularies to describe the same type of objects. It is also common practice to mix terms from different widely used vocabularies with proprietary terms. Thus Linked Data applications need to apply mappings to translate Web data to their local schema before doing any sophisticated data processing. Maintaining a local or central set of mappings that cover all Linked Data sources is likely to be impossible due to the size and dynamics of the Web of Linked Data. Thus we propagate a distributed, pay-as-you-go data integration approach where data publishers, vocabulary maintainers and third parties may publish expressive mappings on the Web. A client application which discovers data that is represented using terms that are unknown to the application may search the Web for mappings and apply the discovered mappings to translate data to its local schema. In order to realize this data integration paradigm, we specify in this design report a language for publishing expressive, named mappings on the Web and a composition method for chaining partial mappings from different sources based on a mapping quality assessment heuristic. In addition to the completely open Web use case, the mapping language can of course also be used to integrate data in more closed enterprise settings.
# Table of Contents

1. **INTRODUCTION** ........................................................................................................................... 7

2. **SPECIFICATION OF THE R2R MAPPING LANGUAGE** .................................................................. 10
   - 2.1 **R2R MAPPINGS** .................................................................................................................... 12
   - 2.2 **SOURCE PATTERN** ................................................................................................................ 16
   - 2.3 **TARGET PATTERN AND MODIFIERS** .................................................................................. 16
   - 2.4 **TRANSFORMATIONS** ............................................................................................................. 19

3. **THE R2R PUBLISHING VOCABULARY** .......................................................................................... 30
   - 3.1 **PUBLISHING MAPPINGS ON THE WEB** ............................................................................ 30
   - 3.2 **INTERLINKING MAPPINGS WITH VOCABULARY TERMS AND DATASET DESCRIPTIONS** ....... 32

4. **INTEGRATION-AWARE LINKED DATA APPLICATIONS** .............................................................. 36
   - 4.1 **ARCHITECTURE** .................................................................................................................... 36
   - 4.2 **OVERVIEW OF THE DATA TRANSLATION PROCESS** ............................................................ 37

5. **RELATED WORK** ........................................................................................................................ 39

6. **ALGORITHM FOR MAPPING CHAINING** ...................................................................................... 41
   - 6.1 **CONSTRUCTION OF THE MAPPING SEARCH GRAPH** ........................................................... 41
   - 6.2 **QUALITY-BASED CONSTRUCTION OF THE MAPPING CHAINS** ....................................... 43

7. **EVALUATION OF THE MAPPING LANGUAGE EXPRESSIVITY** .................................................. 46

8. **OUTLOOK** .................................................................................................................................... 48

9. **REFERENCES** ............................................................................................................................ 50
List of Figures

FIGURE 1: Data about the movie The Shining published by LinkedMDB, DBpedia and Freebase ........................ 30

FIGURE 2: Two R2R mappings which are published by the DBpedia project as Linked Data. ......................... 31

FIGURE 3: R2R mappings for translating rdf:type dbpedia:Person triples into rdf:type foaf:Person ........... 32

FIGURE 4: Architecture of a Linked Data application that employs R2R mappings to ................................. 36

FIGURE 5: The structure of the mapping search graph of our example mappings ...................................... 42

FIGURE 6: The main loop for computing the mapping chains ................................................................... 44

FIGURE 7: The rating function for the mapping chain with mapping m as root ...................................... 44

FIGURE 8: The composition function that assembles the best mapping chain for the given vocabulary node argument .................................................................................................................. 45

List of Tables

TABLE 1: String functions ........................................................................................................................ 21

TABLE 2: Arithmetic functions ................................................................................................................. 21

TABLE 3: List functions .......................................................................................................................... 22

TABLE 4: XPath functions ....................................................................................................................... 23

TABLE 5: Overview of the R2R features that were required to formulate mappings between DBpedia and datasets that are interlinked with DBpedia .................................................................................................. 47
1. Introduction

The Web of Linked Data [1] has grown considerably over the last three years and covers a wide range of different domains today [2]. Linked Data sources use different vocabularies to represent data about a specific type of object. For instance, DBpedia, Freebase and LinkedMDB all use their own proprietary vocabularies to represent data about movies. GeoNames, LinkedGeoData, and the UK Ordnance Survey all use different terms to refer to the concept Administrative District. For other types of objects, vocabularies have emerged that are used by multiple data sources but usually also not by all data sources that provide data about these objects. For instance, FOAF\(^1\) is widely used to represent data about people. As commonly used vocabularies\(^2\) often do not provide all terms that a data source needs to publish its content on the Web of Linked Data, data sources often mix terms from multiple commonly used vocabularies with proprietary terms.

The resulting heterogeneity is a major obstacle to building useful Linked Data applications and thus to realizing the promise of the Web of Linked Data: To enable applications to work on top of a single global dataspace which allows them to discover and integrate new data sources at runtime.

Translating data from a potentially endless set of Linked Data sources to the target vocabulary that is expected by an application requires a large number of mappings. Maintaining a local or central set of mappings that covers all Linked Data sources is likely to be impossible, or at least very costly. Thus this document propagates a distributed, pay-as-you-go data integration approach: Distributed as we build on data publishers, vocabulary maintainers as well as third parties to publish mappings on the Web; pay-as-you-go [3][5] as Linked Data applications are assumed to display Web Data in a rather un-integrated fashion in the absence of mappings, just as Linked Data browsers like Tabulator or Marbles and Linked Data search engines like Sindice or

\(^1\) http://www.foaf-project.org/

\(^2\) http://esw.w3.org/TaskForces/CommunityProjects/LinkingOpenData/CommonVocabularies
FalconS do today. As more effort is invested over time into generating and publishing mappings on the Web (= pay-as-you-go), Linked Data applications can discover these mappings and use them to further integrate Web data in order to be able to deliver more sophisticated functionality.

This document proposes a language for publishing expressive, named mappings on the Web and a composition method for chaining partial mappings from different sources based on a mapping quality assessment heuristic. The R2R Mapping Language is designed to fulfill the following requirements:

1. **Vocabulary cherry-picking**: As data sources mix terms from different vocabularies, the mapping language has to support fine-grained, self-contained term mappings which can be flexibly combined.

2. **Interlinking and discovery**: Every term mapping must be identified with its own dereferenceable URI in order to enable mappings to be interlinked with RDFS or OWL vocabulary term definitions [7][8] and voID dataset descriptions [18], and to allow client applications to discover and retrieve mappings by following RDF links.

3. **Expressivity**: The language needs to provide for structural transformations in order to overcome differing publishing patterns and for property value transformations, for instance in order to normalize different units of measurement.

4. **Dataset-level and vocabulary-level mappings**: Different data sources use different value formats to represent values of the same property. For instance, they provide a distance either in meters or kilometers or names either as first name family name or family name, first name. Therefore the language must provide for dataset-level mappings as well as for more generic vocabulary-level mappings.

The mapping composition method is designed to fulfill the following requirements:

5. **Term-level composition**: The composition method should apply a best-effort approach to generate executable transformations based on all mappings that have been discovered so far. If no direct mapping is available for a term, the method should compose mappings into a mapping chain.

6. **Mapping quality assessment**: As the quality of the mappings that are published on the Web may vary widely, the method should apply a heuristic to assess the quality of mappings and prefer mappings that are likely to deliver better results.
The design document is structured as follows: Section 2 is the specification of the R2R mapping language. Section 3 is the specification of the R2R publishing vocabulary. Section 4 describes how applications can use the mappings to translate Web data to a target vocabulary. Section 5 gives an overview of related work. Section 6 describes the algorithm for chaining mappings based on quality heuristics. Section 7 presents the evaluation of R2R Mapping Language. Section 8 presents an outlook on publishing and discovering mappings on the Web.
2. Specification of the R2R Mapping Language

The R2R mapping language is designed for publishing vocabulary mappings as Linked Data on the Web. Mappings are therefore represented as RDF and each mapping is assigned its own dereferenceable URI. Similar to the SPARQL query language, the R2R mapping language operates on an RDF semantics level and does not rely on any further assumptions about the semantics of Web data.

The R2R mapping language is a declarative language for describing correspondences between terms from different RDF vocabularies and is meant for RDF data translation between these different representations.

The R2R namespace is http://www4.wiwiss.fu-berlin.de/bizer/r2r/

The main elements of the R2R mapping language are:

- **Mappings** define the correspondences between a vocabulary term from the target vocabulary with terms of the source vocabulary. In other words, they define a fine-grained view in terms of the target vocabulary over terms of the source vocabulary.

- **Target Patterns** define the source structure and vocabulary that is mapped from.

- **Transformations** define how source data values are transformed into target data values. For example by using arithmetic or string operations.

- **Target Patterns and Modifiers** define the target structure and vocabulary that is mapped to.

To give a quick overview on the mapping language, the following example shows how mappings can be expressed:
```sparql
@prefix r2r: <http://www4.wiwiss.fu-berlin.de/bizer/r2r/> .
@prefix p: <http://...> .

# Namespace of the mapping publisher
@prefix p:personClassMapping
  a r2r:Mapping ;
  r2r:sourcePattern "?SUBJ a foaf:Person" ;
  r2r:prefixDefinitions "foaf: <http://xmlns.com/foaf/0.1/> . dbpedia: <http://dbpedia.org/ontology/Person>" ;
  r2r:targetPattern "?SUBJ a dbpedia:Person" .

# Property mapping from foaf:name -> dbpedia:name
@prefix p:foafNamePropertyMapping2
  a r2r:Mapping ;
  r2r:sourcePattern "?SUBJ foaf:name ?o ." ;
  r2r:targetPattern "?SUBJ <http://dbpedia.org/ontology/name> ?o" ;
  r2r:prefixDefinitions "foaf: <http://xmlns.com/foaf/0.1/>" .

# Property Mapping that only transforms the literal value, but leaves that vocabulary untouched. This is a pure instance level mapping.
@prefix p:dateValueMapping1
  a r2r:Mapping ;
  r2r:sourcePattern "?SUBJ dc:date ?o" ; # date given in a non-xsd:dateTime/xsd:date compatible format: yyyy/mm/dd
  r2r:targetPattern "?SUBJ dc:date ?'c'^^xsd:dateTime" ; # make it compatible
  r2r:transformation "c = concat(infixListConcat('-', split('/', ?o)), 'T00:00:00')" ;
```

Basically a R2R mapping constitutes one source pattern, one or more target patterns, optional prefix definition for the qualified URIs used inside the source or target patterns, also optionally a reference to a parent mapping and also optionally one or more transformation definitions.

Here are some notes for the example mappings from above; each part is discussed more deeply in its own section:

- Every mapping is of type r2r:Mapping
- A source pattern is defined in a subset of the SPARQL syntax (WHERE clause). In addition the ?SUBJ variable has to be used for the resource that gets mapped to prevent ambiguity.
- Prefix definitions (e.g. line 12) define the prefixes used in the source and target patterns
- A target pattern contains target triples that are constructed with the use of variables of the source pattern or of tranformation patterns. Properties and classes have to be explicitly specified URIs.
• Transformation patterns generate new values which are assigned to a "fresh" variable (line 29). This variable can be used in a target pattern (line 28). On the right hand side of the equal sign are expressions that may be built out of functions, constants, variables and conditionals.

• There also exist several *modifiers* that can be applied inside of target patterns. They modify the data type, language tag or can transform literal values to URIs and vice versa. In line 28 for example the generated variable ?c is assigned a xsd:dateTime data type by the syntax ?'c'^^xsd:dateTime. For numeric data types, source values are automatically casted to the target datatype.

2.1 R2R Mappings

R2R mappings are the main construct of the R2R mapping language. They constitute self-contained units that represent correspondences between terms of two different vocabularies. In this way each mapping can be executed against a dataset independent of other mappings (over the same vocabularies). Similar to a SPARQL CONSTRUCT clause, a r2r:Mapping has a r2r:sourcePattern and a r2r:targetPattern. Mappings are formulated from a target view. This means that we express how target terms - classes or properties - relate to source terms. So every triple in the target pattern could be viewed independently of the other triples in the target pattern and may be generated by considering merely the source pattern.

The following table lists all the important properties of a R2R mapping:

<table>
<thead>
<tr>
<th>r2r:prefixDefinitions</th>
<th>Prefixes used in the source or target patterns must be defined here. There is no local or base namespace! That is, :someName or someName won't work in a pattern. You can have several of these definitions for one mapping. Prefix definitions are inherited by referenced parent mappings or mapping collections as explained below.</th>
</tr>
</thead>
</table>


Each mapping must have exactly one source pattern. It expresses the pattern of the source vocabulary terms. A subset of the SPARQL syntax/expressivity that is valid in a WHERE-clause is allowed here. Also, in order to make it unambiguous which variable in the source pattern corresponds to the mapped resources, the variable ?SUBJ has to be used. Read the source pattern section for more details.

A mapping must have one or more target patterns. One target pattern comprises a set of triples or paths. Paths are just syntactic sugar to express connected/joined triples. There exist several modifiers to specify the node type, data type and the language of a target object. For further details consider the target pattern section.

Definitions used for transforming property values. See Transformation section for detailed information.

A reference to a parent mapping, which can be used to build on the referenced mapping definition. Mainly used to reduce redundancy: The source pattern of the referenced mapping is joined with this source pattern. Prefix definitions do not have to be repeated.

Mapping Collection

Since you will most likely use prefixed URIs in your mappings it can be cumbersome to repeat prefix-definitions over and over again. Another way to cope with this problem is to define them once for a resource that carries redundant data and make mappings reference this resource. The resource is of type r2r:MappingCollection and is currently only used to carry prefix definitions:

```
p:mapColl1 a r2r:MappingCollection ;
```
This object can be referenced by the `r2r:partOfMappingCollection` property, so the prefix definitions don't have to be repeated over and over again:

```
p:mapping1 a r2r:Mapping ;
r2r:partOfMappingCollection p:mapColl1 .
```

### Mapping examples

In the following we give some examples of class related mappings, which defines how a *class term* of the target vocabulary is expressed in the means of the source vocabulary. Mappings are specified as instances of the class `r2r:Mapping`.

In the following example we have a simple *one-to-one* mapping:

```
@prefix r2r: <http://www4.wiwiss.fu-berlin.de/bizer/r2r/> .
# Namespace of the mapping publisher
@prefix p: <http://...> .
p:dbpediaToFoafPersonMapping a r2r:Mapping ;
r2r:targetPattern "?SUBJ rdf:type foaf:Person" ;
r2r:sourcePattern "?SUBJ rdf:type dbpedia:Person" .
```

This mapping defines that every element of class `foaf:Person` is also an element of class `dbpedia:Person`. The instance variable `?SUBJ` must be used in every source pattern and is reserved for representing the instances that are the focus of the mapping. In the example above it represents all `dbpedia:Person` resources.

As you probably noticed, mappings to a class term always have a target pattern like this:

```
?SUBJ (a | rdf:type) <classURI>
```

Next we show some examples where *property terms* are the target of the mapping.

A property in the target schema can correspond to several properties in the source schema and vice versa. Also for property mappings, the use of value transformations becomes relevant. Transformations express how values from the source and the target pattern relate to each other - also from a target view.

The following examples also show different possibilities of how terms correspond to each other. 1-to-1 correspondences relate single terms from each vocabulary. *Many-to-1*
correspondences relate several terms from the source to one term in the target and 1-to-many the other way round:

```turtle
@prefix r2r: <http://www4.wiwiss.fu-berlin.de/bizer/r2r/> .
@prefix p: <http://...> .

# Simple 1-to-1 property mapping: foaf:name => dbpedia:name. The source representation of the # property value is adopted.
p:oneToOnePropertyMapping a r2r:Mapping ;
r2r:sourcePattern "?SUBJ foaf:name ?o" ;
r2r:targetPattern "?SUBJ dbpedia:name ?o" ;

# A Many-to-1 property mapping. This type always has to use value transformations to make sense.
p:manyToOnePropertyMapping a r2r:Mapping ;
r2r:sourcePattern "?SUBJ foaf:firstName ?f . ?SUBJ foaf:lastName ?l" ;
r2r:targetPattern "?SUBJ dbpedia:name ?n" ;
r2r:transformation "?n = concat(?l, ', ', ?f)" ; # Concatenate the first and last name seperated by a comma+space.

# The opposite way of the above mapping. The target pattern references the transformation result # variables instead of variables from the source pattern.
p:OneToManyPropertyMapping a r2r:Mapping ;
r2r:sourcePattern "?SUBJ dbpedia:name ?n" ;
r2r:targetPattern "?SUBJ foaf:firstName ?f" ;
r2r:targetPattern "?SUBJ foaf:lastName ?l" ;
r2r:transformation "?f = getByIndex(split(', ', ?n), 1)" ; # Concatenate the first and last name seperated by a comma+space.
r2r:transformation "?l = getByIndex(split(', ', ?n), 0)" ; # Concatenate the first and last name seperated by a comma+space.

# A "property mapping" referencing a "class mapping".
# This mapping is valid if executed against DBpedia for example.
# But only for instances that are member of dbpedia:Person/foaf:Person.
# That's why we included the r2r:mappingRef to specify the context in which the mapping is valid.
# Without the reference we would end up with alot of non-Person instances having a foaf:name # property, which would lead to conflicts.
p:labelToNameMapping a r2r:Mapping ;
r2r:sourcePattern "?SUBJ rdfs:label ?o" ;
r2r:prefixDefinition "foaf: <http://xmlns.com/foaf/0.1/>" ;
r2r:mappingRef p:dbpediaToFoafPersonMapping .

# A mapping from a schema "Person livesIn Country" to a schema "Person livesIn City locatedIn Country" # The problem here is that there is no City instance in the source schema, so we have to generate one. # The target pattern also includes a path, where the object in the middle (?city) isn't repeated.
p:pathPropertyMapping a r2r:Mapping ;
r2r:prefixDefinitions "..." ;
r2r:sourcePattern "?SUBJ o1:likesIn ?country" ;
r2r:targetPattern "?SUBJ o2:likesIn ?city o2:locatedIn ?country" ;
r2r:transformation "?city = concat('http://target.org/ontology/cities/unknown?', urlencode(?SUBJ))" ;

# The last line of the previous mapping generates a dummy URI. The Person will be the only inhabitant of this city.
```
2.2 Source Pattern

The source part of the mapping constitutes a descriptive language that is based on SPARQL. The source pattern is matched against Web data and binds values to a set of variables. By having analysed several real use cases - for example the mappings in the evaluation, see section 7 - we reduced the original expressivity of the source patterns to the absolutely necessary. The plan is to extend the language in a use case driven way and only add the language features that are actually needed. The following description of the source pattern part of R2R mappings can thus be seen as a core language. Potential extensions are kept in mind, but are not added till the corresponding "nails" show up in our use cases.

A source pattern has the following structure:

- It consists of several triple patterns
- A triple pattern can include URIs, blank nodes, variables and literals.
- The same variable name or blank node used in different triple patterns must be bound to the same value for a solution.
- The ?SUBJ variable must be present in at least one triple pattern. Among other things it is used to combine several mappings without ambiguity.
- In the property position of a triple pattern only explicit URIs are allowed. This accounts for the fact that mappings are defined between vocabulary terms.
- In particular following language features of SPARQL 1.0 are not (yet) part of Source Patterns:
  - UNION (simple UNIONS can be divided into several mappings as alternative)
  - OPTIONAL
  - FILTER
  - GRAPH (unnecessary)

2.3 Target Pattern and Modifiers

The target pattern is used to produce triples in the target vocabulary. Its structure is quite simple:

- A target pattern is a set of triples and/or paths seperated by '.'
- Triples can contain URIs, blank nodes, variables and literals.
- Variables are not allowed in the predicate position and as the object of rdf:type pattern.
- Variables in the target pattern are also bound either by the source pattern or by a transformation pattern.
- Modifiers can be applied to variables in Object position.

A *path* has the form: `Resource Property`, followed by one or more `Resource Property` parts, and ends with an `Object`. A resource can be a blank node, URI or variable.

Path example:

```text
<...> r2r:targetPattern "?SUBJ dbpedia:spouse ?sp dbpedia:livesIn ?o" .
```

It follows from above that only the variable `?o` can be a literal, all in between must be URI references and thus are automatically converted to URIs. Paths are only syntactic sugar and do not increase expressivity.

### 2.3.1 Constant values and URI references

The following listing shows how constants can be placed in target patterns:

```text
# Use constants in target patterns
<someMapping> r2r:targetPattern
  "?SUBJ rdfs:label 'String constant'" ,
  "?SUBJ rdfs:label '''Also a string constant, where you can include 's'\'\'\'" , # " and "\'\'\' are also possible
  "?SUBJ rdfs:label 'language tagged string'@en " ,
  "?SUBJ n:height 1.76" , # Decimal constant
  "?SUBJ n:height 1.76e0" , # Double constant
  "?SUBJ n:height 1.76"^^xsd:double" , # Double constant
  "?SUBJ n:age 54" , # Integer constant
  "?SUBJ n:hasBeenAccepted true" , # Boolean constant

# Use of URIs in target patterns
<someMapping> r2r:targetPattern
  "?SUBJ rdf:type <http://dbpedia.org/ontology/Person>" ;
  "?SUBJ dbpedia:movement dbpedia:Renaissance" .
```

### 2.3.2 Blank nodes

Although not recommended for several reasons, it is possible to insert blank nodes in target patterns:

```text
r2r:targetPattern "[] foaf:name ?name" ;
```
In the first example a new unique anonymous node will be created for every variable binding of ?name. In the second example, the same blank node label is used in two triple patterns. Both of these placeholders will be replaced with the same blank node for each result binding of the variables in the source pattern. In this example the same blank node will be once in subject position and once in object position of the two generated triples. Square bracket blank node syntax like this [ foaf:name ?name ] is not supported, only empty brackets.

2.3.3 Modifiers for variables

To define data type or language for generated values or modify/add the type and language of values bound in the source pattern, R2R offers several modifiers for variables appearing in a triple pattern. Following modifiers exist:

- **URI modifier**: The variable values become URIs - it only makes sense for the rightmost element, which could be a literal or an URI reference. Usage: Enclose the variable name - not the question mark - in angle brackets, for example ?<varname> instead of ?varname.

- **Literal modifier**: is the opposite of the URI modifier. Usage: Enclose the variable name in apostrophes like this ?'varname' instead of ?varname.

- **Language modifier**: adds a language tag to a string literal. This has to be used in conjunction with the literal modifier. Usage: ?'varname'@en adds an "en" language tag.

- **Datatype modifier**: specifies the datatype of the variable values. This has to be used in conjunction with the literal modifier. Usage: ?'varname'^^Datatype-URI

We present now some examples:

**URI modifier**:

# If for example the object of a triple in the source dataset is a literal containing a HTTP link, you can convert it to an URI resource by enclosing the variable name in angle brackets:

```r2r:targetPattern "?SUBJ foaf:homepage ?<o>" . # The former string value of the source dataset```
Properties cannot be modified:

For a property in the Target Pattern, only explicit URIs are allowed. Something like

```plaintext
"?SUBJ ?p ?o"
```

does not work! Thus no modifiers could apply.

```plaintext
"?SUBJ <http://dbpedia.org/ontology/birthDate> ?o",
"?SUBJ prefix:propertyname ?o".
```

The latter two target patterns show correct uses for

```plaintext
# a property in a target pattern
```

Modifiers for variables in "literal position":

Most alternatives exist for literals in a target pattern, as said before, literals can only be at the rightmost place in a path/triple.

If a variable of the source pattern is used in the target pattern without modifier, the source type or language tag is adopted.

```plaintext
"?SUBJ rdfs:label ?l" # If the source label had a language tag or data type assigned to it, the value in the target dataset will have one, too.
```

Modify values taken from the source dataset or generated values from a transformation definition.

In both cases the lexical values of the input values must have the right format!

Trying to transform the lexical value "seven" to an xsd:double will certainly not work.

But transforming values like "seven" into "7" first and then adding a datatype modifier like xsd:integer will work.

Do a transformation on source values and associate the variable c with the result.

```plaintext
"?SUBJ rdfs:label ?c" , # Thus, this means the same as the latter pattern.
"?SUBJ rdfs:label ?c"^xsd:decimal# Convert the variable values to the xsd:decimal data type. Similarly you can use other data types.
```

Note that data type modifiers influence how numerical values are calculated. Especially if you use the ```xsd:decimal``` data type modifier, all numerical functions of a transformation definition that can handle both decimal or double values will switch to decimal calculation.

2.4 Transformations

2.4.1 Transformation Definitions

Transformations are needed if the value format existing in the source dataset differ from the value format or type you want to have in the target dataset. A transformation is defined by an
(unused) result variable name followed by an equals sign and a transformation expression:

```plaintext
<...> r2r:transformation "?varname = function('some string', otherFunction(...), andAnotherOne(andSoOn(...)))" ; # nested functions
# A numeric expression:
r2r:transformation "?numericResult = 2 * ?x - calculateSomething(?var2)" ;
# a conditional expression
r2r:transformation "?eitherOr = [?x = ?y ? 'either' : 'or']" ;
```

Expressions are constructed out of nestable functions. However, as you can see in the examples there are short forms for arithmetic expressions and conditional expressions. These are converted to function calls when being parsed.

Functions can have 0 to n arguments. They can contain constant values like integers, strings etc., functions or arithmetic expressions (+, -, *, / and parentheses) - in other words, arbitrary expressions. The return value of a function is either a single atomic value (always as string representation) or a list of atomic values.

If a list is given as argument where an atomic value is needed the first element of the list is taken and vice versa a single value is treated as a single element list, if a list is needed.

Here are some more examples of transformation definitions:

```plaintext
<...> r2r:transformation
# Split a string at slashes and put it together with minus signs in between
"?c = infixListConcat('-', split('/', ?o))" ,
# Multiply 643.5 with the value from the variable ?o of a source pattern and add 2 to it
"?d = 2 + 643.5 * ?o" ,
"?e = [getByIndex(split(':', ?url), 0) = 'http' ? '' ] " .
```

For some complete mapping examples with transformations see the R2R Mapping section.

Since transformation expressions can result in a single value or a list of values, their use in a target pattern has the following effects:

- For single values: The value is inserted at the position of the triple/path of the target pattern.
- For list values: For every value of the list a triple/path is generated.

Example:
2.4.2 Built-in Functions

Examples on function usages are given after the complete listing.

Table 1: String functions

<table>
<thead>
<tr>
<th>Function</th>
<th>What it does</th>
<th>Returns List?</th>
</tr>
</thead>
<tbody>
<tr>
<td>join(infix, arg1, arg2, ..., argN)</td>
<td>Concatenates arg1 to argN with the infix string given by the first argument</td>
<td>no</td>
</tr>
<tr>
<td>concat(arg1, arg2, ..., argN)</td>
<td>Returns a string of the concatenated argument values</td>
<td>no</td>
</tr>
<tr>
<td>split(regex, stringarg)</td>
<td>Split the second argument at places matching the regex</td>
<td>yes</td>
</tr>
<tr>
<td>listJoin(infix, list)</td>
<td>Concatenates the values of the list argument with infix inserted inbetween</td>
<td>no</td>
</tr>
<tr>
<td>regexToList(regex, stringarg)</td>
<td>Returns a list of strings as specified by the regex</td>
<td>yes</td>
</tr>
<tr>
<td>replaceAll(thisRegex, withThatString, inThisString)</td>
<td>Replaces all matches of the regex with a string</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 2: Arithmetic functions
<table>
<thead>
<tr>
<th>Function</th>
<th>What it does</th>
<th>Returns List?</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(arg1, arg2, ..., argN) or '+'</td>
<td>Add arg1 to argN</td>
<td>no</td>
</tr>
<tr>
<td>subtract(arg1, arg2, ..., argN) or '-'</td>
<td>Subtract arg2 to argN from arg1</td>
<td>no</td>
</tr>
<tr>
<td>multiply(arg1, arg2, ..., argN) or '*'</td>
<td>Multiply arg1 to argN</td>
<td>no</td>
</tr>
<tr>
<td>divide(arg1, arg2) or '/'</td>
<td>Divide arg1 by arg2</td>
<td>no</td>
</tr>
<tr>
<td>integer(arg)</td>
<td>Convert argument to integer value by taking only the integer number part</td>
<td>no</td>
</tr>
<tr>
<td>mod(arg1, arg2)</td>
<td>returns: arg1 modulo arg2</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 3: List functions

<table>
<thead>
<tr>
<th>Function</th>
<th>What it does</th>
<th>Returns List?</th>
</tr>
</thead>
<tbody>
<tr>
<td>list(arg1, arg2, ..., argN)</td>
<td>Create a list out of the arguments</td>
<td>yes</td>
</tr>
<tr>
<td>sublist(listarg, from, to)</td>
<td>Returns a sub list of the given list argument from index &quot;from&quot; to index &quot;to&quot; (exclusive)</td>
<td>yes</td>
</tr>
<tr>
<td>subListByIndex(listarg, i1, i2, ..., iN)</td>
<td>Build a list from the given list, but with elements picked as specified by the index arguments</td>
<td>yes</td>
</tr>
<tr>
<td>listConcat(listArg1, listArg2, ..., listArg3)</td>
<td>Concatenate the list arguments to one list</td>
<td>yes</td>
</tr>
</tbody>
</table>
### XPath functions

These are functions mirroring string and numeric XPath functions\(^3\). The following table does only explain differences, for a function documentation we link to the respective XPath documentation.

<table>
<thead>
<tr>
<th>Function</th>
<th>What it does</th>
<th>Returns List?</th>
</tr>
</thead>
<tbody>
<tr>
<td>xpath:abs(x)</td>
<td>Returns the absolute value of the argument.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:ceiling(x)</td>
<td>Returns the smallest number with no fractional part that is greater than or equal to the argument.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:floor(x)</td>
<td>Returns the largest number with no fractional part that is less than or equal to the argument.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:round(x)</td>
<td>Rounds to the nearest number with no fractional part.</td>
<td>no</td>
</tr>
</tbody>
</table>

---

\(^3\) [http://www.w3.org/TR/xpath-functions/#string-functions](http://www.w3.org/TR/xpath-functions/#string-functions), [http://www.w3.org/TR/xpath-functions/#numeric-value-functions](http://www.w3.org/TR/xpath-functions/#numeric-value-functions)
<table>
<thead>
<tr>
<th>Function</th>
<th>What it does</th>
<th>Returns List?</th>
</tr>
</thead>
<tbody>
<tr>
<td>xpath:round-half-to-even(x)</td>
<td>Takes a number and a precision and returns a number rounded to the given precision. If the fractional part is exactly half, the result is the number whose least significant digit is even.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:codepoints-to-string(cp1, ...)</td>
<td>Creates an xs:string from a sequence of Unicode code points.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:string-to-codepoints(str)</td>
<td>Returns the sequence of Unicode code points that constitute an xs:string.</td>
<td>yes</td>
</tr>
<tr>
<td>xpath:compare(s1, s2)</td>
<td>Returns -1, 0, or 1, depending on whether the value of the first argument is respectively less than, equal to, or greater than the value of the second argument, according to the rules of the collation that is used.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:compare(s1, s2, collation)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xpath:codepoint-equal(s1, s2)</td>
<td>Returns true if the two arguments are equal using the Unicode code point collation.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:concat(s1, ...)</td>
<td>Concatenates two or more arguments to a string.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:string-join((s1, ...))</td>
<td>Returns the string produced by concatenating a sequence of strings using an optional separator.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:string-join((s1, ...), separator)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xpath:substring(s, start)</td>
<td>Returns the string located at a specified place within an argument string.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:substring(s, start, length)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xpath:string-length(s)</td>
<td>Returns the length of the argument.</td>
<td>no</td>
</tr>
<tr>
<td>Function</td>
<td>What it does</td>
<td>Returns List?</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>xpath:normalize-space(s)</td>
<td>Returns the whitespace-normalized value of the argument.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:normalize-unicode(s)</td>
<td>Returns the normalized value of the first argument in the normalization form specified by the second (optional) argument. Note: Implemented normalization forms are NFC (default), NFD, NFKC and NFKD.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:normalize-unicode(s, norm)</td>
<td></td>
<td>no</td>
</tr>
<tr>
<td>xpath:upper-case(s)</td>
<td>Returns the upper-cased value of the argument.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:lower-case(s)</td>
<td>Returns the lower-cased value of the argument.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:translate(s, map, trans)</td>
<td>Returns the first string argument with occurrences of characters contained in the second argument replaced by the character at the corresponding position in the third argument.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:encode-for-uri(s)</td>
<td>Returns the string argument with certain characters escaped to enable the resulting string to be used as a path segment in a URI.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:iri-to-uri(s)</td>
<td>Returns the string argument with certain characters escaped to enable the resulting string to be used as (part of) a URI.</td>
<td>no</td>
</tr>
<tr>
<td>xpath:escape-html-uri(s)</td>
<td>Returns the string argument with certain characters escaped in the manner that html user agents handle attribute values that expect URIs. Note: This is not working correctly to the specification. Try to avoid the function.</td>
<td>no</td>
</tr>
<tr>
<td>Function</td>
<td>What it does</td>
<td>Returns List?</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td><code>xpath:contains(s, c)</code></td>
<td>Indicates whether one string contains another string.</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>*Note: Other than in the XPath function, a collation must not be specified.</td>
<td></td>
</tr>
<tr>
<td><code>xpath:starts-with(s, c)</code></td>
<td>Indicates whether the value of one string begins with another string.</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>*Note: Other than in the XPath function, a collation must not be specified.</td>
<td></td>
</tr>
<tr>
<td><code>xpath:ends-with(s, c)</code></td>
<td>Indicates whether the value of one string ends with another string.</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>*Note: Other than in the XPath function, a collation must not be specified.</td>
<td></td>
</tr>
<tr>
<td><code>xpath:substring-before(s, c)</code></td>
<td>Returns the string that precedes in that string another string.</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>*Note: Other than in the XPath function, a collation must not be specified.</td>
<td></td>
</tr>
<tr>
<td><code>xpath:substring-after(s, c)</code></td>
<td>Returns the string that follow in that string another string.</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>*Note: Other than in the XPath function, a collation must not be specified.</td>
<td></td>
</tr>
</tbody>
</table>
### Function

<table>
<thead>
<tr>
<th>Function</th>
<th>What it does</th>
<th>Returns</th>
<th>List?</th>
</tr>
</thead>
</table>
| xpath:matches(s, pattern) | Returns an boolean value that indicates whether the value of the first argument is matched by the regular expression that is the value of the second argument.  
*Note: There may be differences to the XPath regular expression syntax. If in doubt, consult the Java regex syntax.* | no      |      |
| xpath:replace(s, pattern, replacement) | Returns the value of the first argument with every substring matched by the regular expression that is the value of the second argument replaced by the replacement string that is the value of the third argument.  
*Note: There may be differences to the XPath regular expression syntax. If in doubt, consult the Java regex syntax.* | no      |      |
| xpath:tokenize(s, pattern) | Returns a sequence of one or more strings whose values are substrings of the value of the first argument separated by substrings that match the regular expression that is the value of the second argument.  
*Note: There may be differences to the XPath regular expression syntax. If in doubt, consult the Java regex syntax.* | yes     |      |

### Examples of function usage:

All examples represent string values of the `r2r:transformation` property.

String functions:
# Join several strings by ' - ': ('2010', '07', '01') => '2010-07-01'

?joinedString = join('-', ?year, ?month, ?day)

# Concatenate several strings:
?concatenatedString = concat('Hello ', 'World')

# Split a date: '2010-07-01' => ['2010', '07', '01'] and join the list again
?sameAsDate = listJoin('-', split('-', ?date))

# Do the same as with the split function in the previous example,
# but with regexToList
?interestingStrings = regexToList('(.*)-(.*)-(.*)', ?date)

# Replace all occurrences of '-' by '/' in the date string
?formattedDate = replaceAll('-', '/', ?date)

**Arithmetic functions:**

# Basic arithmetic operators

?six = subtract(multiply(add(1, 2, 3), divide(3, 2)), 1, 2)

or shorter

?six = (1+2+3)*(3/2) - 1 - 2

# Truncate the non-integral part. The following gives 3

?intPart = integer(10/3)

# Get the modulo. The following returns 1

?rest = mod(10, 3)

# modulo also works with floating point numbers, the following returns
# circa 0.1 (not exact, because of floating point arithmetic!)

# If you modified the transformation variable in the target pattern like this:
# ?'rest'^^xsd:decimal you get exactly 0.1

?rest = mod(10, 3.3)

**List functions:**

*Note that the index of the first element of a list is 0.*

# Create a list out of single arguments

list(1, 2, 3) => [1, 2, 3]

# Get the second and third element of a list: [1, 2, 3, 4] => [2, 3]

subList(list(1, 2, 3, 4), 1, 3) => [2, 3]

# Construct a list from another list given the indexes: [1, 2, 3, 4, 5] => [4, 2]

subListByIndex(list(1, 2, 3, 4, 5), 3, 1) => [4, 2]

# Concatenate lists to one list ([1, 2, 3], [4, 5]) => [1, 2, 3, 4, 5]

listConcat(list(1, 2, 3), list(4, 5)) => [1, 2, 3, 4, 5]

# Get a single element of a list [1, 2, 3] => 3

getByIndex(list(1, 2, 3), 2) => 3

# Length of a list [1, 2] => 2

length(list(1, 2)) => 2

length(1) => 1

**2.4.3 Conditionals**
There is also an **If-Then-Else** like construct. The syntax is:

```
[ expression CompOp expression ? expression : expression ]
```

where `expression` can be anything that's allowed on the right hand side of the equal sign of a transformation definition and `CondOp` is one of the following comparison operators: `<`, `<=`, `=`, `>=`, `>`, `!=`

Example:

```...
... r2r:transformation "?varname = concat([?sex = 'female' ? 'Ms.' : 'Mr.'], ?lastname)"
```

### 2.4.4 External Function Loading

It is possible to associate mappings with external Function resources (located in the classpath, filesystem, remote URL etc.). Functions have to be provided as a Java class, so most major languages that compile to Java byte code should be able to implement the necessary Interfaces `Function` and `FunctionFactory`. To use an external function in a mapping you can refer to it like this:

```r2r
:mapping a r2r:Mapping ;
r2r:importFunction "functionName = http://domain/functions/function1" ;
r2r:transformation "?varName = functionName(...)"
```

This will import the function referenced by the URI right of the equal sign under the supplied function name. In the transformation definitions for this mapping, the function can then be called with the function name.

A function resource has the following properties:

```r2r
:function1 a r2r:TransformationFunction ;
r2r:codeLocation <http://domain/functioncode/function1.jar> ;
r2r:qualifiedClassName "de.fuberlin.wiwiss.r2r.external.FunctionFactory1" .
```

- code location specifies where the class code is located. It can be a jar-file or a directory code base. The latter is expanded automatically with the qualified class name to get the full URL to the class file. In the example a Jar file is fetched from an URL.
- The qualified class name specifies a class that implements the `FunctionFactory` interface, which manages the creation of Function objects of that particular type, which will then be executed on the input data.
3. The R2R Publishing Vocabulary

In order to enable Linked Data applications to discover mappings on the Web, R2R mappings can be interlinked with RDFS or OWL term definitions as well as with voiD dataset descriptions[18]. This enables vocabulary or dataset publisher as well as third parties to point applications in the right direction in their search for appropriate mappings.

3.1 Publishing Mappings on the Web

This subsection introduces the R2R Publishing Vocabulary and explains how mappings are interlinked with RDFS and OWL vocabulary term definitions as well as with voiD dataset descriptions. The Publishing Vocabulary is explained along the use case of integrating data about movies from DBpedia4, Freebase5 and LinkedMDB6. The listing below shows a subset of the data provided by these sources for the movie The Shining.

```
01: # Data from LinkedMDB
03:  linkedmdb:director
04:       <http://data.linkedmdb.org/resource/director/8476> ;
05:  linkedmdb:runtime "146" .
06:
07: # Data from DBpedia
08: <http://dbpedia.org/resource/The_Shining_%28film%29> a dbpedia:Film
09:  dbpedia-owl:director dbpedia:Stanley_Kubrick ;
11:
12: # Data from Freebase
13: <rdf:freebase.com/ns/guid.9202a8c04000641f80000000046c3da>
14:  freebase:film.film.directed_by freebase:en.stanley_kubrick ;
15:  freebase:film.film.runtime freebase:m.0k6ftd .
16:  freebase:m.0k6ftd
17:  freebase:film.film_cut.runtime "146.0"^^xsd:float .
```

**Figure 1:** Data about the movie The Shining published by LinkedMDB, DBpedia and Freebase (namespace declarations are omitted).

The representation of information varies between all three sources. The datatypes, lexical values, property and class names and even the graph structure (Freebase: runtime) are

---

4 http://dbpedia.org/about

5 http://www.freebase.com

6 http://www.linkedmdb.com
The following mappings will factor in all these differences. The next listing contains two R2R mappings which are published as Linked Data by DBpedia and which map the `linkedmdb:director` property to the `dbpedia-owl:director` property and the `freebase:runtime` property to the `dbpedia-owl:runtime` property. As can be seen a value transformation is applied in line 18 and a data type modifier is applied in lines 17.

```
01: <http://mappings.dbpedia.org/r2r/LinkedMDBDirectorToDirector>
02: rdf:type r2r:Mapping ;
04:  lmdb: <http://data.linkedmdb.org/resource/movie/>" ;
05: r2r:sourcePattern "?SUBJ lmdb:director ?director" ;
06: r2r:targetPattern "?SUBJ dbpedia-owl:director ?director" ;
07: dc:creator www4:is-group/resource/persons/Person4 ;
08: dc:date "2010-06-23"^^xsd:date .
09: 
10: <http://mappings.dbpedia.org/r2r/FreebaseFilmRuntimeToRuntime>
11: rdf:type r2r:Mapping ;
16: r2r:targetPattern "?SUBJ dbpedia-owl:runtime
17:  ?'runtimeInSeconds'^^xsd:double" ;
18: r2r:transformation "?runtimeInSeconds = ?runtimeInMinutes * 60" ;
19: r2r:sourceDataset <http://mappings.dbpedia.org/r2r/freebaseVOID> ;
23: 
24: <http://dbpedia.org/ontology/runtime> r2r:hasMapping
25: <http://mappings.dbpedia.org/r2r/FreebaseFilmRuntimeToRuntime>.
```

Figure 2: Two R2R mappings which are published by the DBpedia project as Linked Data.

The last listing for our example contains a mapping between the terms `dbpedia-owl:Person` and `foaf:Person` and a mapping between the `linkedmdb:runtime` property and the `freebase:runtime` property. The mappings are assumed to be published on the Web by a third party. To map to the more complex structure that is present in Freebase in line 18 a new resource (URI) is generated to which the actual runtime literal is attached (lines 16, 17).

```
01: <http://thirdparty.org/mappingDbpediaPersonToFoafPerson>
02: rdf:type r2r:Mapping ;
04:  foaf: <http://xmlns.com/foaf/0.1/>" ;
05: r2r:sourcePattern "?SUBJ dbpedia-owl:Person" ;
06: r2r:targetPattern "?SUBJ rdf:type foaf:Person" ;
07: dc:creator <http://thirdparty.org/andreas> ;
08: dc:date "2010-06-11"^^xsd:date .
09: 
10: <http://thirdparty.org/mappingRuntimeLinkedmdbToFreebase>
11: rdf:type r2r:Mapping ;
12: r2r:prefixDefinitions
14:  fb: <http://rdf.freebase.com/ns/>" ;
15: r2r:sourcePattern "?SUBJ movie:runtime ?runtime" ;
16: r2r:targetPattern "?SUBJ fb:film.film.runtime ?generatedURI
17:  fb:film.film_cut.runtime ?'runtime'^^xsd:float" ;
18: r2r:transformation "?generatedURI = concat(?SUBJ, "Runtime")" ;
19: r2r:sourceDataset <http://mappings.dbpedia.org/r2r/linkedmdbVOID> ;
20: r2r:targetDataset <http://mappings.dbpedia.org/r2r/freebaseVOID> ;
21: dc:creator <http://thirdparty.org/andreas> ;
```
3.2 Interlinking Mappings with Vocabulary Terms and Dataset Descriptions

In order to enable Linked Data applications to discover mappings on the Web of Linked Data, R2R mappings are interlinked with RDFS or OWL term definitions that are published according to the best-practices provided by Berrueta and Phipps[10] as well as with voiD dataset descriptions[18]. The R2R mapping language defines the link type r2r:hasMapping to interlink mappings with RDFS or OWL term definitions and voiD dataset descriptions. Lines 24 and 25 contain a r2r:hasMapping link pointing from the vocabulary term dbpedia-owl:runtime to the FreebaseFilmRuntimeToRuntime mapping. Linked Data applications that dereference the vocabulary term dbpedia-owl:runtime receive this r2r:hasMapping link together with the definition of the term and can follow it to discover the mapping.

We will now describe the all parts of the mapping vocabulary. In addition to the typically used properties to describe resources like

- rdfs:label
- rdfs:comment
- dc:date
- dc:creator

we define properties that in addition to their informative benefit are actually used for finding mappings on the Web or to chain mappings in our discovery process. These properties are:

- r2r:mapsTo: Links from the mapping to the target vocabulary terms (properties or classes).
- r2r:dependsOn: Links to the source vocabulary terms this mapping depends on.
- r2r:sourceDataset: Specifies the data source - if any - this mapping can be executed
on, or rather it states that the data it can be run on must comply with the assumptions on the data of the specified data source.

- **r2r:targetDataset**: Specifies the target data source - if any - that the data produced by this mapping complies to.

- **r2r:hasMapping**: This property can be used by vocabulary or dataset publishers to link to mappings that are relevant to them. It should be used together with vocabulary term resources of RDFS or OWL vocabularies or dataset resources given as voID descriptions.

With these links on hand a linked data crawler like ldspider\(^7\) can now find mappings by following links from vocabularies to mappings and the other way round. The same applies for voID dataset descriptions. This will enable us to find mappings that are linked by vocabulary or dataset publisher. To find mappings from third parties search engines like Sindice\(^8\) can be used. Queries like "list all mappings that map to foaf:name" can easily be stated with these search engines to find mapping resources.

### 3.2.1 Dataset Level and Vocabulary Level Mappings in more Detail

The r2r:sourceDataset and r2r:targetDataset properties mentioned before may be a bit confusing, that's why we explain their importance for the Web publishing use case in greater depth.

As different data sources use different value formats to represent values of the same RDF property, the R2R Mapping Language distinguishes between vocabulary-level mappings and dataset-level mappings. Vocabulary-level mappings are usually more generic and might be applied to transform data from and to all data sources that use a specific vocabulary term. Dataset-level mappings specify how data should be translated between two specific data sources. They usually define more detailed transformations to overcome property value heterogeneity, for instance by normalizing different units of measurement or by adding language tags or data types to property

\(^7\) [http://code.google.com/ldspider](http://code.google.com/ldspider)

\(^8\) [http://sindice.com](http://sindice.com)
values. Mixed cases are also possible, where instead of two data sources, the source or the target is actually on vocabulary level.

To make the point clear, we give examples for each of the four possible combinations: 1. No dataset specified, 2. both source and target dataset specified, 3. only target dataset or 4. only source dataset specified. Let’s suppose there are two vocabularies vocA and vocB and both vocabularies define name, lastName and firstName properties. The name property in both vocabularies is not constrained any further other than that it must be a string describing the name of a person. Now, there is also a data source dsA using the property vocA:name, but consistently uses a specific format for this property - for example "last name, first name" - and a dataset dsB with vocB:name, but with a different format - "first name last name". Then the four possible combinations are:

1. Mapping from vocA to vocB - a pure vocabulary level mapping
   - Example: Map vocA:name to vocB:name

2. Mapping from vocA / dsA to vocB / dsB - a pure dataset level mapping
   - Example: Map vocA:name to vocB:name property with value transformation: 'last name, first name' → 'first name last name' (split and concat)

3. Mapping from vocB to vocA / dsA - vocabulary on the source side, data source on the target side
   - Example: Map (vocB:firstName, vocB:lastName) to vocA:name with value transformation: concat(?lastName, ', ', ?firstName)

4. Mapping from vocA / dsA to vocB - data source on the source side, vocabulary on the target side
   - Example: Map vocA:name to vocB:firstName and vocB:lastName: 'last name, first name' → 'last name' or 'first name'

So, if a mapping applies property value transformations or structural transformations that only make sense on specific input data, the mapping publisher can restrict the scope of the mapping to be used only with input data that conforms to the publishing pattern of a specific data source by adding a r2r:sourceDataset triple to the mapping pointing at the voID dataset description of the data source. This means that the mapping makes assumptions about the input data that go
beyond the specification of the source vocabulary term.

Similarly, if a mapping applies value transformations or structural transformations to produce data according to the publishing pattern of a specific data source, the mapping publisher can annotate the mapping to produce specific output by interlinking it with the voID description of the target dataset using the `r2r:targetDataset` link type.

The main benefits of distinguishing dataset level and vocabulary level mappings are:

1. You can express correspondences between vocabularies that were not possible by looking at the vocabulary specifications alone. Example: Let's assume you want to map from foaf:name to a `vocX:name` property that exactly specifies the value format. The only way to do this and to end up with values that always comply with the `vocX:name` specification is to state the source dataset you map from and that you know the value format of. There can't be a pure vocabulary level mapping in this case!

2. You can not only ask for data given in a specific target vocabulary, but even more specific, to which value format or unit of measurement etc. the output data will conform to.

3. The third point actually builds on top of point 1. The discovery process (Section 4.2) benefits from the increase in possible mapping paths, which can now alternate between the dataset level and vocabulary level. Example: `foaf:name → (vocA:firstName, vocA:lastName) → vocB:name`, here the first mapping needs to specify `r2r:sourceDataset` (format needs to be known for extracting first and last name) and the second mapping is a pure vocabulary mapping, but can also define a `r2r:targetDataset` hint if the produced values conform to some dataset.
4. Integration-aware Linked Data Applications

This section describes how R2R mappings are used by Linked Data applications to translate Web data to an application-specific target schema. Here we describe the architecture of the application. For usage details consult section 5.6 of this document.

4.1 Architecture

The goal of the application is to extract data from various Web datasets according to a given definition of the target data. This definition is given by a target vocabulary (see 5.6.1 for details) that your own application can understand and work with. By automatically choosing and applying R2R mappings to transform the original data, the target output is eventually achieved and can then be used by your application.

Below, we give an overview of a possible architecture of a Linked Data application that works on top of the open Web of Linked Data. Within closed enterprise use cases, the application architecture can differ from the one discussed.

![Architecture of a Linked Data application that employs R2R mappings to translate Web data to its local target schema.](image)
The architecture consists of a *Web Data Access Module* which retrieves RDF data from the Web by following RDF links. The access module stores Web data in a *Temporal Store*. The data is represented as a set of Named Graphs where all data from one data source is contained in its own Named Graph. These graphs are called dataset graphs. If available the URI of the corresponding voiD description is attached to each dataset graph. R2R mappings that are discovered on the Web are stored in a *Mapping Repository*. In addition to discovering mappings by following RDF links, the Web Data Access Module also queries the Semantic Web Search engines Sindice and FalconS for further R2R mappings. This ensures that third party mappings are also discovered. The application provides the *R2R Mapping Engine* with a description of the target vocabulary. The description consists of a simple set of URIs identifying the terms (properties as well as classes) of the target vocabulary. The mapping engine translates the data from the temporal store into the target vocabulary and stores the resulting triples in the target repository. Afterwards, it deletes the data in the temporal store. The application can now issue queries using the target vocabulary against the target repository.

If the application wants property values to have a specific format, for instance *last name, first name* or a distance being given in kilometers, it can instruct the mapping engine to produce only triples in this format by annotating the corresponding target vocabulary term with the URI of the voiD description of a data source that provides exactly this value format. The mapping engine will then only consider mappings having this voiD description as *r2r:targetDataset* annotation as the last element of the mapping chain and will thus only produce property values having this format.

### 4.2 Overview of the Data Translation Process

The R2R Mapping Engine applies a mapping composition method for selecting and chaining partial mappings from different sources based on a mapping quality assessment heuristic. The mapping chains that are most likely to produce high-quality output data are then employed to translate data to the target vocabulary. The complete mapping composition and data translation process is described in section 6. In the following, we give a brief overview of the process:

1. For each DSGn, the engine determines the set of vocabulary terms (SVTn) that are used within the graph.
2. For each term TVTn in TVT and each DSGn, the engine builds a mapping search graph which contains the mappings that can be chained to connect TVTn to term in SVTn.
3. The engine constructs the mapping chain MCTVTnDSGn from the search graph which is most likely to produce high quality translations based on the mapping quality assessment heuristics described below.

4. The engine executes each mapping chain and writes the resulting triples into the target repository.

The quality of any content that is published on the Web is uncertain due to the open nature of the Web [23]. In the Linked Data context uncertainty applies to instance data, links between data sources as well as to mappings. Thus before mappings should be used, a Linked Data application needs to assess their quality and decide whether it wants to trust discovered mappings. The mapping quality assessment heuristic used by the R2R Mapping Engine is based on the following assumptions:

1. As we expect the quality of vocabulary-level mappings provided by vocabulary maintainers themselves to be higher than the quality of mappings provided by third parties, the engine prefers mappings which are published in the same domain as the vocabulary itself.

2. As we expect the quality of dataset-level mappings provided by data publishers to be higher than the quality of mappings provided by third parties, the engine prefers mappings which are published in the same domain as the target dataset, afterwards it considers mappings published in the same domain as the source dataset, and then mapping published in any domain.

3. Because every mapping is a potential source of failure, we expect the quality of data translations to decrease with the length of the mapping chains. The mapping engine therefore prefers short mapping chains wherever possible. The details on how these heuristics are applied within the mapping quality assessment function used by the R2R Mapping Engine are described in Section 6.
5. Related Work

There is a large body of related work on ontology alignment in the knowledge representation community [20][11] as well as a large body of related work on schema matching in the database community [21]. Both communities have also developed formalisms to represent correspondences in the form of mappings. For this document, we consider the task of generating correspondences out of scope and focus solely on publishing existing correspondences on the Web and on discovering and combining mappings from different sources.

The idea of using SPARQL to express vocabulary mappings has been proposed by Polleres [16]. The authors describe a variation of SPARQL called SPARQL++, which enhances the expressivity of SPARQL in a way so that more classes of vocabulary mappings can be expressed. Schenk [25] also uses SPARQL to define views over data sources and graphs. Both languages are designed for use cases where the set of mappings is closed and known in advance. In contrast, R2R focuses on the use case of publishing and discovering mappings on the Web.

For publishing mappings on the Web, the RDF Schema [7], OWL [8], and SKOS [9] recommendations provide the terms rdfs:subClassOf / rdfs:subPropertyOf, owl:equivalentClass / owl:equivalentProperty, as well as the SKOS mapping properties. These constructs can be used to represent simple term correspondences. But they are not expressive enough to represent more complex correspondences which involve structural transformations or property value transformations like normalizing different units of measurement. They are thus not sufficient to achieve deeper integration. More expressive mapping languages for RDF data have been developed by Euzenat [19] and Haslhofer [22]. These languages provide a high expressivity but do not provide for interlinking mappings with other Web resources. Thus, they do not allow applications to discover mappings in a follow-your-nose fashion [13] as the R2R Framework does.

This document builds on the dataspace paradigm that is being developed within the database community [3]. We adopt a pay-as-you-go data integration approach [3][5], as for Web-scale data integration it does not make sense to apply a schema-first integration approach which relies on a unifying schema to be modelled over all data sources before the dataspace can be used. Thus we consider a step-by-step integration approach, which decreases heterogeneity over time, as more
likely to succeed [5]. Existing pay-as-you-go data integration systems [4], like PayGo, iMeMex or SEMEX, assume that a single authority controls a dataspace and that this authority also administrates the mappings that are used to incrementally decrease heterogeneity. In contrast, by building on mappings that are published and interlinked on the Web, we propagate a distributed, community-based approach to mapping provision. A dataspace system that builds on similarly fine-grained term mappings as the approach presented in the paper is iTails [11]. In contrast to our work which uses mappings for data translation, iTails uses mappings (hints) for query expansion.
6. Algorithm for Mapping Chaining

This section is about building mapping chaining. In a functional sense it can be seen as “nesting” mapping executions. Alternatively, if you understand mappings as view definitions, one view is defined on other views and so on until the leaf views are defined on the actual source vocabulary. The first subsection describes the search graph and how it is initially built. The second subsection then pays attention on a selection algorithm for picking to the select a mapping chain based on quality heuristics.

6.1 Construction of the Mapping Search Graph

This section describes how the mapping search graph is built. We want to compute the mapping chain \( M_{TVTnDSGn} \) for a given dataset graph \( DSGn \) and the target vocabulary term \( (TVT_n) dbpedia:runtime \). However, the mapping may depend on vocabulary terms that are used in its source pattern which are not elements of the source vocabulary terms \( SVT_n \).

Within our example mapping these are \( fb:film.film.runtime \) and \( fb:film.film_cut.runtime \), which would then constitute sub-goals that need to be resolved. Figure 5 shows how our search graph for the example mappings looks like.
There are vocabulary nodes and mapping nodes. A mapping node relates to one individual mapping. A vocabulary node relates to a vocabulary term and optionally a dataset. Vocabulary nodes pool mapping nodes by the vocabulary terms that the mappings produce. If mappings specify a \texttt{r2r:targetDataset}, they only pool those mapping nodes whose target dataset is the same. In Figure 2 the vocabulary node \texttt{(movie:runtime, dataset:LinkedMDB)} would be such a dataset specific node. This means that a mapping to \texttt{movie:runtime} that additionally defines LinkedMDB as target dataset would point to both \texttt{movie:runtime} nodes in Figure 2. Which of the two types of vocabulary nodes a mapping node has to pick to satisfy each of its dependencies is determined by the source dataset dependence of the mapping. The mapping from Figure 3 for example defines \texttt{LinkedMDB} as its source dataset restriction. Thus the only vocabulary node that can satisfy this dependency is \texttt{(movie:runtime, dataset:linkedMDB)}. The reason for this is that the mapping in Figure 3 makes assumptions about the input data that go beyond the specification of the vocabulary term. But these assumptions are probably not met by the values produced by the mappings of the dataset unspecific vocabulary node. The property specification of \texttt{movie:runtime} does not mention the measurement unit of its values. Still, in the LinkedMDB dataset the values are represented in minutes. Mappings to \texttt{movie:runtime} that comply to this assumption can therefore state \texttt{LinkedMDB} as target dataset and thus are the only valid candidates to be chained.
with the mapping from Figure 3.

We build the search graph by following sub-goals down to a specified depth. Our main goal in this step is to find a set of vocabulary terms from $SVT_n$. These become what we will call source nodes $SN_i$, which are special vocabulary nodes that constitute satisfied subgoals. Then in a second step, mappings are chained in a forward manner starting from this subset of $SVT_n$, which will be described in more detail in 6.2. In Figure 5 the vocabulary node $(movie:runtime, dataset:linkedMDB)$ is such a source node for the input dataset LinkedMDB.

In the following, we formalize the search graph $G = (V, E)$, where $V$ is the set of nodes and $E$ a set of directed edges. The node set $V$ can be split into two disjoint sub-sets $V = VM \cup VV$, the set $VM$ of mapping nodes and the set $VV$ of vocabulary nodes. A node $n_i \in VV$ has the following properties: It is defined by a unique combination of a vocabulary term $VT_i$ and a target dataset. The target dataset can be unspecified for dataset unspecific nodes. For each mapping node $m$ that maps to this combination there exists an edge $(m, n_i)$ in $E$. Furthermore the node $n_i \in VV$ has a directed edge to the mappings that depend on this exact combination. For each $VT_i$ there is also one extra node $n_e$ with edges $(m, n_e)$ in $E$ for all the mapping nodes $m$ that map to $VT_i$, but do not define a target dataset plus for every mapping node of the dataset specific ones. This complies with the discussion of valid mapping chains from above: A dataset level mapping – which specifies a source dataset – can only be chained to mappings that specify that target dataset. A vocabulary level mapping on the other hand – which specifies no source dataset - can chain both target dataset specific and unspecific mappings to its dependencies.

### 6.2 Quality-based Construction of the Mapping Chains

The quality of any content that is published on the Web is uncertain due to the open nature of the Web. In the Linked Data context uncertainty applies to instance data, links between data sources as well as to mappings. Thus before mappings should be used, a Linked Data application needs to assess their quality and decide whether it wants to trust discovered mappings. The mapping quality assessment heuristic used by the R2R Mapping Engine is based on the assumptions discussed in Chapter 4.2:

In the following, we explain how the mapping chain $MC_{TVTnDSGn}$, that is most likely to produce
a high quality translation according to the assumptions given above, is built from the search graph. The main idea of the rating algorithm is to have several iterations, starting in the first iteration with the set S of source nodes SNi which all get a best-rating of 1. All other nodes get a best rating of 0. In each iteration i the best MC of depth i will be calculated for each node, that is, every path from the root to a leaf has at most a length of i. For example in iteration 1 the best MCs that consist of just one mapping which can be directly executed against DSGn are found. Figure 6 contains the main loop for computing the mapping chains.

```plaintext
01 For i=1 to maxIteration:
02   Set S_new = {}
03   For each node e ∈ S:
04     for each m ∈ e.dependentMappings:
05       rate(m)
06       for n ∈ m.associatedNodes:
07         if (m.score > n.bestScore)
08           // set m to best mapping of n and update n.bestScore
09           update(n, m) and S_new.add(n)
10   S = S_new
```

**Figure 6: The main loop for computing the mapping chains**

The idea is to (re-)rate the mapping chain with mapping M_i as root when a DEPi of M_i increased its best score. M_i is thus automatically chained with the best MC found at DEP_i so far. The rating function is given in Figure 7. The construction of MC_{TVTnDSGn} is finally done with the compose function given in Figure 8.

Once the mapping chain MC_{TVTnDSGn} has been constructed, it can be used to translate data from DSG_n to the target vocabulary term TVT_n. In this process the data from DSG_n is piped through the mapping chain. The leaf nodes of MC_{TVTnDSGn} are fed directly from the DSG_n, and intermediate nodes by a combination of DSG_n data and child node output. The data is propagated up to the root of MC_{TVTnDSGn}, where the actual TVT_n triples are produced.

```plaintext
01 rate(m):
02   score = rateFactors(m) // compute score of the individual mapping
03   for(Node dep ∈ m.dependencies):
04     score = score * dep.bestScore
05   return score
```

**Figure 7: The rating function for the mapping chain with mapping m as root**

```plaintext
01 compose(node):
02   m = node.bestMapping // Init a mapping composition with m as root
03   MappingComposition mc = mappingCompositionInit(m)
04   for(Node dep ∈ m.dependencies):
```

**Figure 8: The compose function for MC_{TVTnDSGn}**
// recursively call compose on the mapping's dependencies (nodes)
05   MappingComposition depC = compose(dep)
// add mapping composition depC of dep to the mapping chain mc
06   addSubtree(mc, dep, depC)
07 return mc

Figure 8: The composition function that assembles the best mapping chain for the given vocabulary node argument
7. Evaluation of the Mapping Language Expressivity

We have tested the expressivity of the R2R Mapping Language by formulating mappings between DBpedia and all data sources that are interlinked with DBpedia\(^9\). The mappings will be published as Linked Data on the Web as part of the next DBpedia release (before the workshop). In addition all mappings can be downloaded as a single file from http://mappings.dbpedia.org/r2r/DBpediaToX.n3. Table 5 gives an overview of the R2R features that were used to formulate the mappings. The abbreviations in the table headings refer to the following R2R features that were required for formulating the mappings: \textit{URI replace} = Simple translation by replacing URIs of the source vocabulary with URIs of the target vocabulary; \textit{Struct trans 1:1} = Structural transformation of the RDF graph describing an instance; \textit{Struct trans 1:n} = Structural transformation where one instance in the source dataset results in the creation of multiple instances in the target dataset; \textit{Val trans} = Literal value transformation for instance using string function; \textit{UoM trans} = Unit of measurement normalization; \textit{DT mod} = Data type modifier applied to literal value; \textit{LG mod} = Language modifier applied to literal value; \textit{L2U mod} = Modifier applied to create a URI from a literal value.

<table>
<thead>
<tr>
<th>Class : Data sources</th>
<th>URI replace</th>
<th>Struc trans 1:1</th>
<th>Struc trans 1:n</th>
<th>Val trans</th>
<th>UoM trans</th>
<th>DT mod</th>
<th>LG mod</th>
<th>L2U mod</th>
</tr>
</thead>
<tbody>
<tr>
<td>Place : GeoNames / DBpedia</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Artist : MusicBrainz / DBpedia</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Place : NYT / DBpedia</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(9\) http://wiki.dbpedia.org/Interlinking
The R2R Mapping Language proved to be expressive enough in this experiment to represent all mappings that were required to translate data between the representations used by the sources. The experiment also showed that far more expressivity is required to properly translate data to a target schema than currently provided by standard terms such as owl:equivalentClass / owl:equivalentProperty [8] or rdfs:subClassOf / rdfs:subPropertyOf [7].

<table>
<thead>
<tr>
<th>Source</th>
<th>DBpedia</th>
<th>Factbook</th>
<th>BookMashup</th>
<th>Gutenberg</th>
<th>US Census</th>
<th>Dailymed</th>
<th>Linkedmdb</th>
<th>Drugbank</th>
<th>Freebase</th>
<th>Musician</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Book</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Author</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>County</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organiza.</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Film</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drug</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Film</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Musician</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Overview of the R2R features that were required to formulate mappings between DBpedia and datasets that are interlinked with DBpedia
8. Outlook

A key difference that distinguishes Linked Data technologies from other approaches to sharing data on the Web, like Web 2.0 APIs, is that the Linked Data principles enable data to be published in a self-descriptive manner on the Web and thus ease the integration of data from different sources. An important aspect of self-descriptiveness is the reuse of terms from common vocabularies that are made accessible on the Web via dereferenceable URIs and thus allow client applications to retrieve term.

But as common vocabularies often do not cover all aspects of published data and as choosing terms from common vocabularies increases the effort involved in Linked Data publication, we experience that a lot of data on the Web is represented using proprietary terms.

This document takes the self-descriptiveness of Web data a step further by proposing a framework to enable data publishers, vocabulary maintainers and third parties to publish fine-grained vocabulary mappings on the Web and to interlink these mappings with other web resources so that they can be discovered by client applications.

This approach decomposes the Web-scale data integration problem along two dimensions: Time and effort allocation. Data integration can be realized in a pay-as-you-go fashion over time, meaning that data providers can follow Tim Berners-Lee’s raw data now practice and start publishing Linked Data using any vocabulary. Later, they (or third parties) can invest effort into reusing terms from common vocabularies and/or invest effort into creating mappings which tie the data to related data sources. This lowers the entry barrier for participating in the Web of Linked Data to a large extent. For applications, this means that they can initially provide only basic services on top of very large dataspace. As more mappings become available in the dataspace, larger parts of the dataspace can be integrated more deeply, and the quality of the provided services can be increased accordingly. The second dimension is effort allocation. In classic data integration settings as well as within the context of Web 2.0 mashups, the data integration effort is solely shouldered by the data consumer. By publishing mappings on the Web, data publishers and third parties - like industry consortia - may provide integration hints. This enables the data integration effort to be split between data publishers, third parties and the data consumer.
As the amount of instance data on the Web of Linked Data is growing rapidly and as more and more data providers start setting RDF links between instances, we think that it is time for the Linked Data research community to focus its attention on overcoming the vocabulary-level heterogeneity which we observe on the Web of Linked Data. We see the R2R Framework as a first step into this direction and hope that R2R will interest further groups to work on solutions for integrating Web data based on mappings that are provided by various parties to a public dataspace and which are thus inherently of uncertain quality. For Linked Data publishers, it is also the next logical step to increase the self-descriptiveness of their data by starting to publish vocabulary mappings for proprietary terms. They could already start today with publishing coarse-grained mappings in the form of `owl:equivalentClass`, `owl:equivalentProperty`, `rdfs:subClassOf`, `rdfs:subPropertyOf` links. As more consensus develops in the community around mapping languages for representing richer correspondences, publishers could switch to these languages following the pay-as-you-go paradigm.
9. References


