Automatic transformation between XML documents with different namespaces

Victor Porton

Aug 28, 2014 04:43:43

1 Draft status

Moved to https://en.wikiversity.org/wiki/Automatic_transformation_of_XML_namespaces – so now this document may be outdates, see the Wikiversity site instead.

This is a preliminary rough draft. There may be errors, omissions, and logical inconsistency. There should be added more formal explanations (not just examples as now), more examples, and more explanation. The standard may change.

See more about this recommendation at http://freesoft.portonvictor.org/namespaces.xml.

Please comment.

TODO: Say that if some transformation was already used, then use *the same script* not just the same transformation when requested.

TODO: Say if that a script is invalid, do not consider the entire RDF file invalid, but just ignore this script.

2 General notes

What is the semantic of an XML namespace? It may mean a relation of XML code with things of the real world. But a part of semantic which can be formally specified for an XML namespace, is the rules of transformation between this namespace and other XML namespaces.

This standard specifies the formal properties of semantic for an XML namespace, particularly the rules of transformations between different XML namespaces. It also contains means to validate an XML file given its namespace, even when it contains multiple namespaces.

The aim of this specification to support current practices of using namespaces. It however may also direct namespace creators and users in the future, to be compatible with this specification.

For no ambiguity of terminology, I will call any kind of RDF resources (on the Web or elsewhere) just as *RDF file*.

3 Relations to other standards

Resource Directory Description Language (RDDL) may be linked from our RDF by the `:link` element with `:role="`http://www.rddl.org/`"`. The meaning of `:nature` and `:purpose` attributes is described by Associating Resources with Namespaces, this is incorporated into this specification by reference. RDDL specification advises to place an RDDL document at the namespace URL. This is discouraged by this specification which advises to put an RDF document conforming to this specification at the namespace URL. It is recommended that validation is described by means of this standard, not backward compatible `:link` element.

4 Rationale

The Web is a mixture of standard and nonstandard resources. There were huge rumors that the Web would be enhanced by XML technology into something radically new, based on semantic rather than markup. In large these hopes failed. The Web continues to be dominated by semi-markup rather than purely semantic file formats, including HTML and XHTML.

For a real radical change for the better, we need to develop a mean to automatically transform between XML namespaces.

This specification is the specification of how to do these transformations.

The list of particular rationale points of this specification:

- mixed namespace documents without the need to specify a special relationship between every pair of namespaces
- semantic markup on the Web, with anybody (if he is a programmer) being able to create his own markup extensions; this may be a transitional (by converting to XHTML markup) stage of getting rid of old HTML and replacing it with semantic markup; new markup may be especially usefully created by search engine owners
- cross-browser markup extensions
- creating “composite” markup languages
- no need to download transformers and so easy upgrades
- possibility to create a browser which internally understands only XSL FO (and/or SVG)

This is based on RDF because RDF is effective on describing relationships between entities (and XML namespace are entities).

This standard may be implemented as:

- a command line utility;
- a part of an interactive Web browser;
- as an HTTP proxy server.

5 Implementation considerations

The term *sandboxing* means running an untrusted program in such a way that it is unable to damage system or user files or disclose sensitive information. Reliable sandboxing is required for all implementations of this standard, which may access network, as otherwise the implementation would be potentially harmful by running malicious program loaded from the Web. Implementation note: http://portonsoft.wordpress.com/2014/01/11/toward-robust-linux-sandbox/

As on certain systems it is too hard or impossible to implement sandboxing for secure dealing with converter programs downloaded from the Web, it is feasible to implement this recommendation as HTTP proxy servers on supporting systems, so that user's of non-supporting system would access the functionality related with this standard through these proxies.

Implementation of this recommendation in the form of a HTTP proxy server is also useful for using it with non-supporting Web browsers (including all browsers shipped before this standard is written).

6 Our namespace

We use RDF (RDF/XML, Turtle, or an other RDF format) as our data format.

The examples in this specification are in Turtle format.

In the examples we assume the following Turtle directives:

```
@prefix : http://portonvictor.org/ns/trans/ . # update the namespace in a final
version of this document
@prefix dc: http://purl.org/dc/terms/ .
```

7 User options

Upon user options are specified:

1. the source XML document;
2. one or more destination namespaces;
3. a (possibly empty) set of RDF files *(RDF files which are downloaded before main loop)*;
4. a (possibly empty) set of transformers; (These transformers are used even if their precedence is lower than of an other possible transformer.)
5. recursive retrieval order: an ordered subset of the three elements set {*see also*, *sources*, *targets*};
6. additional transformers which are run before and after the main loop.

If recursive download is on, it also should be specified whether to search depth-first or breadth-first.

The user specifies whether to prefer a transformer in the RDF describing the source namespace or in the RDF describing the target namespace. This influences only what (and in what order) RDF resources are loaded. After an RDF is loaded it is subject to "first loaded" rule below.

RDFs loaded earlier take precedence over RDFs loaded later. (Note: This facilitates *uniformity* of transformations and validations in the sense that different transformers and validation rules are not applied to the same namespaces.)

8 RDF grammar

The below describes grammar and meaning of an RDF file, as it should be used in our software.

After reading an RDF file, some information (namespaces, transformations, etc.) is extracted from it, as the RDF file is considered as an instance of the below defined grammar.

Remark: The below grammar always applies to one RDF files, it never applies to several RDF files merged.

Rationale: If the data were merged on loading every new RDF file, this would possibly create contradictory (not conforming to the grammar described below) information when loading the next RDF file.

Cardinality of a predicate P for a subject S is the number of objects for these predicate and subject. We will denote cardinalities as:

- **0..1** (a number equal to 0 or 1)
- **1..1** (exactly 1)
- **0..*** (any natural number)

- **1..*** (any positive natural number)

In this document *root objects* are `:namespace` and `:transformer`.

For each RDF node for which it is stated that it belongs to a root object, we will require that certain predicates on this node have certain cardinality, that nodes of certain objects of these nodes have certain cardinality, etc.

I will denote this as a tree: The root node is a class URL. The rest nodes are interleaving predicates together with cardinalities and nodes.

I will write additional requirements for a node in parentheses. Specifically if a node must be of an XML Scheme datatype, I will write the type in brackets.

I will call such trees *format trees*.

Note that subjects may have additional predicates not described in the format trees. For example (below) objects of class `:namespace` may have predicates `:link`.

TODO: Define what "conforms to grammar tree" means.

RDF file is *valid* (for the purposes of this specification) if it conforms to grammar tree and all additional validity constraints defined in this specification.

See below for examples.

Extracted RDF data will be named in this specification as indicated by italicized text in parentheses.

8.1 Extracting information from RDF

Every time an RDF file is loaded, the following algorithm takes place: [TODO: Order of extracting information?]

1. Store all triples with predicates `rdfs:subClassOf` and `:higher-than` and `:lower-than`.
2. process `rdfs:seeAlso` as described below.
3. For every of the below defined RDF classes (`:namespace`, `:transformer`) find *start nodes* that is RDF nodes which have `rdf:type` predicate with the given class as object.
4. For every start node verify that its objects linked with the predicates specified in the grammar trees have the cardinalities specified by the grammar trees (do this recursively) and do all other validation checks.
5. If a tree for a start node is not valid, it is an error. The error should be handled as specified below.
6. Reject information from start nodes which violates any validation rules. Extract namespaces, transformations, and precedences as get from the rest (that is valid) start nodes and their grammar trees.

When processing scripts (for validation of a namespace or for a transformer), just ignore scripts of unknown `rdf:type`. Process scripts of below defined types just like (TODO: explain what is "just like") start nodes.

9 RDF resource format

9.1 `rdfs:seeAlso` predicates

When reading an RDF file, it should process triples of the forms:

- `:validation rdfs:seeAlso` *URL*
- `:transformation rdfs:seeAlso` *URL*
- `:all rdfs:seeAlso` *URL*

This should add the URL to the list of RDF files to be downloaded (in the order of recursive retrieval described elsewhere in this specification). If we are doing validation, add it only when the subject is `:validation` or `:all`. If we are doing transformation, add it only when the subject is `:transformation` or `:all`.

Rationale: We may want the same transformation in source and target namespaces. So we want do split it into a separate file loaded by `rdfs:seeAlso` directive.

TODO: It seems that I completely forgotten to describe validation algorithm!

9.2 Scripts

A *script* is something which accepts an input (some XML text, in this specification) and generates an output (a text and/or a program exit status). (A script may be a Unix command, Web service, etc.)

A script is represented as an RDF node with certain properties.

This specification provides the following classes of scripts:

- command line
- script in a specified programming language
- A Web service

TODO: Check XML syntax after every step.

9.2.1 Command line

- `:command-line`
  - {1..1} `:command-string` [`xsd:string`] *(command line)*
  - {0..1} `:validate-subdocuments` [`xsd:boolean`] *(whether to validate subdocuments)*

Example:

```
:script
      a :command-line ;
      :command-string "xmllint --format" .
```

In this example, XML is just validated and reformatted, not really modified.

9.2.2 Script in a specified programming language

- `:named-script`
  - {1..1} `:language` (IRI) *(programming language)*
  - {0..1} `:min-version` *(minimum version)*

- {0..1} `:max-version` *(maximum version)*
- {1..1} `:script-url` *(URL of the script)*
- {0..1} `:ok-result` *(result denoting OK)*
- {0..1} `:completeness` *(completeness)*
- {0..1} `:stability` *(stability)*
- {0..1} `:preference` *(preference)*
- {0..1} `:validate-subdocuments` [`xsd:boolean`] *(whether to validate subdocuments)*

9.2.3 A Web service

- `:web-service`
  - {1..1} `:form` (URI) *(request URI)*
  - {1..1} `:method` *(HTTP method)*
  - {1..1} `:xml-field` [`xsd:string`] *(field for XML)*
  - {0..1} `:validate-subdocuments` [`xsd:boolean`] *(whether to validate subdocuments)*

Validity constraint: `:validate-subdocuments` should be present only for validators.

9.3 RDF describing a namespace

Namespaces are described as instances of `:namespace` class.

Their format tree:

- `:namespace`
  - {0..*} `:validator` *(validator)*
    - _ (script node)

Example:

```
<http://purl.org/dc/terms/>
      a :namespace ;
      dc:description <http://...> ;
      # Other Dublin Core metadata.
      :link [
            :url <http://www.rddl.org/> ;
            :role <http://www.rddl.org/> ;
            :nature <http://www.w3.org/1999/xhtml> ;
            :purpose <http://www.rddl.org/purposes#schema-validation>
      ] ;
      :validator [
            a :named-script ;
            :language <http://portonvictor.org/ns/trans/scripts/#Python> ;
            :min-version "2.1" ;
            :max-version "3.2" ;
            :script-url <http://example.org/script.py> ;
            :ok-result "OK" ;
            completeness 0.9 ;
            stability 0.9 ;
            preference 0.9 ;
            :validate-subdocuments true
      ] ;
```

A `:validator` is specified in the same way as `:script-data` (see below), except that `:transformer-kind` parameter is ignored. The validator may have `:ok-result` to specify what output of the validator signifies a valid document. In absence of `:ok-result` for `:named-script` and `:command-line` valid

document is signified by successful command return value (0 on Unix) and for `:web-service` the value of `:ok-result` defaults to empty string.

In `:validator` the property `:language` may also refer to a namespace URL of some XML scheme (such as http://www.w3.org/2001/XMLSchema). In this case `:ok-result` is ignored.

A human readable description of a namespace should be specified with Dublin Core parameters.

The `:link` nodes are like resources in RDDL (but with our namespace instead of RDDL namespace).

A namespace description may provide `:validate` parameter to specify how to validate the documents whose root element is of our namespace. The `:validate` parameter has a subparameter `:nature` which should be understood accordingly RDDL specification.

If `:validate-subdocuments="true"` is specified, every subdocument indicated by a primary node, which is not a child of an other primary node, is validated using its namespace, if available. Example: This works well for SVG or MathML documents embedded into an XHTML document. It should give a warning if in document there are namespaces which cannot be validated.

There may be multiple `:validate` parameters in order to allow to use schema of different natures.

`:link` parameter with subparameters `:role` and `:nature` is backward compatible with RDDL and should be understood in accordance with the RDDL specification.

9.4 RDF describing a transformer

Note: Transformers should be run in a secure sandbox, so that they would be unable to damage or read user's files. Also the time of the entire operation should be limited. (Rationale: If we are going to limit particular parts of the entire process rather it as a whole, then we would be unable to limit parts of operations done by sandboxed application, and the entire stuff would make no sense.) We may also limit the total amount of data transferred through the network, if the operating system supports it. (We can't limit a specific operation inside the sandbox.)

Implementation note: Such sandboxing can be implemented for example with SELinux for Linux. It is tempting to use Java security manager, but as of start of 2014 year, Java security is too buggy and therefore should not be used.

Their formal tree:

- `:transformer`
  - {1..1} `:source-namespace` *(source namespace)*
  - {0..1} `:target-namespace` *(target namespace)*
  - {1..1} `:precedence` *(precedence)*
  - {0..*} `:script` *(script)*
    - _ (script node)

Here is an example of an XSLT transformer:

```
<...>
      a :transformer ;
      dc:description <http://...> ;
      # Other Dublin Core metadata.
      :source-namespace <...> ;
      :target-namespace <...> ;
```

```
:precedence <...> ;
:script [
      a :xslt ;
      :version "2.0" ;
      :script-url <http://example.org/scripts/foo.xslt> ;
      :transformer-kind :entire ;
      :argument [
            :name "debug" ;
            :value false
      ] ;
      :argument [
            :name "other" ;
            :value 123
      ] ;
      #:initial-context-node ... ; # See XSLT 2.0 spec.
      initial-template "first" ;
      initial-mode: "first" ;
      completeness 0.9 ;
      stability 0.9 ;
      preference 0.9
] .
```

Both `:source-namespace` and `:target-namespace` parameters are not required.

It is recommended but not required that objects of predicates `:source-namespace` and `:target-namespace` are of `:namespace` class.

A transformer may have no target NS. Example: XInclude. In this case every NS in consideration can act as the target.

We need to define precedences for different kinds of transformers, for example we would probably have the precedence "include" for XInclude and other cross-document facilities, "macro" for macroses, or precedence "formatting" for a transformer generating XSL formatting objects or SVG.

9.4.1 Common arguments

All transformers are subclasses of the class `:transformer`. All transformers accept the following parameters:

- `:transformer-kind` may be `:entire`, `:sequential`, `:up-down`, `:down-up`. It is used accordingly the section "Order kinds of of document transformers".
- `:completeness`, `:stability` and `:preference` specify a number 0..1.0. `:completeness` describes how much of the transformer is implemented. `:stability` describes how reliable is the transformer (that is whether it is likely to crash or produce meaningless results), `:preference` is to denote other factors for calculating priority (see below).

Priority of a chain of transformations is calculated using completeness, stability, and preference of the links of the chain. The recommended algorithm is to multiply all completenesses, stabilities, and precedences of all links and then sum them.

9.4.2 Particular types of transformers

XSLT

`:xslt` is a subclass of `:script`. It additionally accepts the following parameters:

- `:version` the version of XSLT.

- `:script-url` the URL of the XSLT script
- `:arguments` (optional) parameters with name `:name` and a value `:value` specifies arguments for the XSLT processor. [TODO: `:arguments` → `:argument`]
- `:initial-template` (optional) specifies the initial template for XSLT
- `:initial-mode` (optional) specifies the initial mode for XSLT

Java, Python, Ruby, et al

```
:script [
      a :named-script ;
      :language <http://portonvictor.org/ns/trans/scripts/#Python> ;
      :min-version "2.1" ;
      :max-version "3.2" ;
      :script-url <http://example.org/script.py>
]
```

This example means that the script http://example.org/script.py is run by Python interpreter of at least 2.1 up to 3.2 version.

Recommendation: If several suitable versions of the interpreter are available, use the maximal allowed version.

The following languages should be available:

- Python
- Java
- Ruby
- Perl
- TODO

Web service.

```
:script
      a :web-service ;
      :form <http://example.org/form> ;
      :method "post" # or "get" ;
      :xml-field "text" .
```

This sends POST request to http://example.org/form which should return an XML document.

9.5 Describing precedences

A precedence is an RDF-S class. (It's members are transformers.)

```
<http://example.org/precedences/macro>
      rdfs:subClassOf <...> ;
      :higher-than <...> ;
      :higher-than <...> ;
      :lower-than <...> .
```

The predicates `:higher-than` and `:lower-than` can apply to transformers as well as to classes of transformers.

The following rules are used to deduce which entities have “higher than” precedence relative an other entity:

- If `:higher-than` parameter is specified inside a `:precedence` description then the described

entity is of higher precedence than the referred to entity.

- If `:lower-than` parameter is specified inside a `:precedence` description then the referred to entity is of higher precedence than the described entity.
- If an entity A has higher precedence than an entity B and the entity B has higher precedence than an entity C, then the entity A has higher precedence than the entity C.
- If a class C has higher precedence than an entity B and A is an entity of class C then A has higher precedence than B.
- If an entity A has higher precedence than a class C and B is an entity of class C then A has higher precedence than B.

The entities are related by "higher than" relation if and only if this relation can be deduced from the above rules (for all currently loaded RDF resources).

If a circle of precedences is encountered this is a fatal error.

10 Recursive downloading the information

When we speak about *downloading next RDF file* we mean downloading the next RDF file in the depth-first or breadth-first order (as specified by user options). The order of edges which point to the next URL for the downloading algorithms are: first "see also", "sources", "targets" (in the order specified by the user options) and then the order of their start nodes in the RDF file.

Upon reading an RDF file, the list of the namespaces, the list of transformations, and the list of relations between precedences should be updated, accordingly the grammar described above.

Implementation note: If our task is validation, updating the list of transformations and precedences is not necessary. If our task is transformation, updating the list of namespaces is not necessary.

Note: Retrieving some existing documents designated by namespace URLs gets an RDF Scheme. Examples: http://www.w3.org/1999/02/22-rdf-syntax-ns# and http://purl.org/dc/elements/1.1/. In this case `:namespace-info` in the RDFS points to a namespace information which should be downloaded instead.

As an other alternative, the RDF file pointed by the namespace URL may contain namespace and transformer info.

Several namespaces may refer to one RDF due using # in URLs.

Backward compatible feature: If at the URL we download from there is an RDDL document, download the RDF specified as an RDDL resource with xlink:role="http://portonvictor.org/ns/trans/". (This is not done recursively, that it an RDDL document linked from an other RDDL document is not downloaded.)

If a valid RDF document was not retrieved from a URL (say because of a 404 HTTP error or because invalid XML code), it should not be retrieved again. However it is not forbidden to repeat failed because of timeout HTTP attempts.

Recursive retrieval may be limited to certain URLs. For example, it may be limited only to file:// URLs.

11 Transformations

11.1 Variables used by the algorithm

The following data is kept during processing the main loop:

- the list of loaded RDF files;
- list of used transformers (a mapping from pairs of namespaces into transformers). [TODO: What's about transformers without target namespace?]
- current XML document;
- the list of namespaces, whose info has been loaded;
- the list of transformations loaded;
- the list of transformations performed;
- the precedences data.

The list of available transformations between two given namespaces is:

- one element list consisting of the transformation, if this transformation was performed;
- the list of all transformations with given source and target.

It is *ready for transformation* when and only when there is a path from every namespace in the current document ending either in a destination namespace or in no namespace.

11.2 Figuring out the next transformer

If a fixed list of transformers is given by the user, the next transformer is just the next in the list.

If there is no fixed list of transformers, then:

1. Chose a transformer among first transformers of the available chains with the highest precedence ending with either the destination namespace or no namespace, among the available transformers having as the source one of current document namespaces. Transformers having the same source and target should be skipped. (Note: Such transformers nevertheless may be useful as user-specified transformers.)
2. If there are several transformers of the same precedence, choose the transformer for which there exists an available chain with highest priority. Note that only the first element of the chain is actually used, the rest are for calculation of priorities only.
3. Add the chosen transformer to the list of used transformers.

If there is choice of transformers of equal precedence, the application may give a warning.

A sequence of transformation is built in order to elaborate a transformation in the best way. This may be done taking into account completeness and stability of transformers. One possible formula which may (but is not required) be used is to choose a path for which the product of all completeness and stability values in the path is maximal. (If a transformation in the chain duplicates has the same source and target as an earlier used transformation, then use the earlier used transformation. Rationale: Keep consistency.)

The chosen sequence of transformers must have no loops.

If it is unable to determine the next transformer in the precedence order, then the processor should either give an error, or give a warning and choose the order arbitrarily.

11.3 Transformation process

Transformation consists of applying to the source document every transformer in turn.

Transformers just get an XML representation on input and output an XML representation.

These namespaces for which transformers are not specified remain intact. (See below for the exact algorithm.)

After the transformation is finished, at user option, fail if there are namespaces not in the destination list. At user option, erase all tags (and their descendants) not in the destination list.

11.3.1 Order kinds of of document transformers

For up-down and down-up transformer (see below) we first we select *primary nodes* from the set of document nodes. The primary nodes are:

1. the root element node;
2. element nodes whose parent has a different namespace than their parent;
3. element nodes which have a namespaced attribute of a different namespace than the element itself.

An element is *having* a certain namespace, if it is either an element of this namespace or has an attribute of this namespace.

Rationale: This twist about namespaced attributes is useful for such usages as XLink. Example:

```
<myElement xmlns:xlink="http://www.w3.org/1999/xlink">...</myElement>
```

may transform into

```
<a xmlns="http://www.w3.org/1999/xhtml"
href="http://www.w3.org/1999/xlink"><myElement>...</myElement></a>.
```

In this example the link is “raised”, above the element it is applied to, to a “wrapper” element (`<a>` in this example).

Next stage, feed every primary node, having a namespace which is a source namespace of the transformer, to the transformer. This constructs a new document.

After this the stage is repeated.

Note: This facilitates for example embedding a complete XHTML document into our XML. The document transformer will receive only the root of the XHTML not its fragments which it may not understand.

The entire document transformer

The entire document transformer is just applying once the transformer to the entire document.

Sequential transformer

The transformer is applied to the first element having the source NS. After this the step is repeated until there are no elements of the source namespace.

Note for implementers: You can start searching for the next element from the point fed to the last transformer.

It should give a warning (or error?) if the head element of the transformed subtree remains with the same namespace. May we skip to the next element in order to avoid an infinite loop?

Up-down transformer

Find all elements of the source NS which are not descendants of an element of the source NS.

Run the transformer for every subtree designated by these elements.

Repeat these steps until there are no elements of the source namespace.

Down-up transformer

Find all primary elements which have no primary descendants. Transform every subtree designated by these elements.

Then find the set of primary elements which are the first primary ancestor of these elements, and replace the set of elements in consideration with these elements.

Repeat the steps until the set it empty.

11.4 The main loop

The main loop consist of repeatedly:

1. applying all available transformations;
2. downloading next RDF file.

If a transformer was once used, then use it even if its precedence or priority is lower than of a new transformer. (Rationale: It is necessary to keep consistency of meanings of XML tags, so that for example an XML tag meaning emphasis would not be sometimes translated into bold and sometimes into italic text.)

Rationale: The following algorithm allows not to download the destination namespace RDF at all, in the case if user requires to load destination namespaces last and the transformation happens to succeed without loading destination namespace. It also can do the reverse thing (that is load only destination namespace and don't load source namespaces).

Before the main loop the user specified list of RDF resources is downloaded. Then all user-specified additional transformers which should be run before the main loop are run.

TODO: In which order to run transformations?

If we are ready for transformation do all transformations (in order defined by precedence, as described in “Figuring out the next transformer”), otherwise download the next RDF file. Do the previous sentence in a loop. Stop if all namespaces in the document are in the destination namespaces list, or we are not ready for transformation and there are no more RDF for download.

Then all user-specified additional transformers which should be run after the main loop are run.

Rationale: Requiring all namespaces in the current document (not only for the root element) allows to apply precedences (what is important for example for correct XInclude processing).

12 Validation

First, RDF files which are downloaded before main loop are downloaded.

Then ???

13 Error handling

If an RDF triple is erroneous, the whole RDF file should be skipped (not loaded). TODO: Make this feature optional.

If a required RDF triple is missing for some bundle [TODO: define "bundle"] only this bundle is skipped, not the entire RDF resource. Rationale: This is because additional information may be loaded from other RDF files, such as additional files specified by the user to be loaded. [TODO: So?]

TODO: Specify what exactly are errors.

For some RDF predicates it is specified (TODO: the list of all such predicates) that they may have no more than one value (that is no more than one RDF object) per RDF subject.

If there are duplicate values for such predicates inside one RDF file, it is an error.

14 RDF-S and OWL schema

RDF-S and OWL for our format are put at http://portonvictor.org/misc/namespaces.rdf (a preliminary not complete scheme).

15 Examples

Down-up may be used with validating parsers or other parsers which may "eat" nodes of different namespaces.

Need to develop test cases.

16 Future directions

Specifying a transformer with source/target being *several* namespaces treated as if it would be one NS. (Example: Dublin Core and `dcterms:` namespaces.)

Non-XML output formats. (For these only entire document transformers can be applied.) We can also use non-XML input formats.

XProc: An XML Pipeline Language

Can we process several namespaces at once when the transformations (with different source namespaces) have exactly equal precedence? (Hm, this cannot be done if the transformers have different kinds. Should we enable concurrent processing of several namespaces when *both* their precedence and their order kind is the same?) Should we point one or several processors (one for each NS) for these multiple-namespaces transformers? (The transformers should be of the same order kind.)

The option of interactive choosing order of transformers.

It should be customizable for a user. It can be done as a user-specified set of RDF files. (Note that values in user-specified files should take precedence over other RDF files.)

There should be a (finite or infinite) mapping from a URL to several URLs when we downloading them.

Should we introduce "composite" scripts (consisting of several transformations sequentially)? First, they would badly interact with searching transformation path. Second, it looks like a cart ahead of horse that in this case we define a script through transformations (not vice versa). Mentioning this, are there weighty enough arguments to add such a construct?

17 The moderated site with transformations

It may be not always possible to put needed RDF files at namespace URLs.

As such, we should create a site to which everyone would be able to upload their connections from URLs (of namespaces) to RDF resources.

It should be a user option whether first load RDF resources from the namespace URL or from this site.

Note that this site must be moderated, as otherwise it would be a temptation for hackers to put their evil transformations for namespaces which are popular in the Web.

We need to create software and a non-profit organization to manage this site.