Noname manuscript No.
(will be inserted by the editor)

X3ML Mapping Framework for Information Integration in Cultural Heritage and beyond

Yannis Marketakis · Nikos Minadakis · Haridimos Kondylakis · Konstantina Konsolaki · Georgios Samaritakis · Maria Theodoridou · Giorgos Flouris · Martin Doerr

Received: date / Accepted: date

Abstract The aggregation of heterogeneous data from different institutions in cultural heritage and e-science has the potential to create rich data resources useful for a range of different purposes, from research to education and public interests. In this paper, we present the X3ML framework, a framework for information integration that handles effectively and efficiently the steps involved in schema mapping, Uniform Resource Identifier (URI) definition and generation, data transformation, provision and aggregation. The framework is based on the *X3ML mapping definition language* for describing both schema mappings and URI generation policies and has a lot of advantages when compared to other relevant frameworks. We describe the architecture of the framework as well as details on the various available components. Usability aspects are discussed and performance metrics are demonstrated. The high impact of our work is verified via the increasing number of international projects that adopt and use this framework.

1 Introduction

Managing heterogeneous data is a challenge for cultural heritage institutions, such as archives, libraries, and museums, but equally for research institutes of descriptive sciences such as earth sciences [51], biodiversity [61, 60], clinical studies and e-Health [47,46]. These institutions host and maintain various collections with heterogeneous material, usually stored in relational databases and often described by different metadata schemata. For exploiting this wealth of data and handling these metadata as a unified set is vital in many applications, including information retrieval, data integration [43], data migration [45] and creating rich domain-specific applications. In this direction, complex query and integration mechanisms have to be designed and implemented to enable uniform access to heterogeneous and autonomous data sources [44].

In order to allow data transformation and aggregation, it is required to produce mappings, to relate equivalent concepts or relationships from the source schemata to the aggregation schema, i.e. the target schema, in a way that facts described in terms of the source schema can automatically be translated into descriptions in terms of the target schema (also known as the "enterprise model" [28]). This is the mapping definition process and the output of this task is the mapping, i.e., a collection of mapping rules.

In this paper we describe the *X3ML mapping framework* designed to support information integration for resource discovery. In such a scenario, alternative knowledge violating cardinality constraints is equally relevant for resource discovery search. Thus, the current version of the framework does not take into account cardinality constraints or other rules in the target side. It guarantees only strict inheritance and subsumption and does not enforce any other constraints.

The framework is able to support the data aggregation process by providing mechanisms of data transformation and URI generation. Mappings are specified using the *X3ML mapping definition language*, a declarative, human readable language that supports the cognitive process of a mapping. Unlike XSLT, that is intended to be comprehensible only by IT technicians, the *X3ML mapping definition language* can be understood by non-technical people as well. Thus, a domain expert is capable of both verifying the semantics, and reading and validating the schema matching. This model

Institute of Computer Science, FORTH-ICS, Greece
E-mail: {marketak, minadakn, kondylak, konsolak, samarita, maria, fgeo, martin}@ics.forth.gr

carefully distinguishes between mapping activities carried out by the domain experts, who know and provide the data, and from the IT technicians, who actually implement data translation and integration solutions. *X3ML mapping framework* serves as an interface between them.

Usually, schema matching is used to describe the process of identifying that two different concepts are semantically related. This allows the definition of the appropriate mappings that are being used as input for the transformation process. However, a common problem is that the IT experts do not fully understand the semantics of the schema matching and the domain experts do not understand how to use the technical solutions. For this reason, in our approach the schema matching and the URI generation processes are separated. The schema matching can be fully performed by the domain expert and the URI generation by the IT expert, therefore solving the bottleneck that requires the IT expert to fully understand the mapping. Furthermore, this approach keeps the schema mappings between different systems harmonized since their definitions do not change, in contrast to the URIs that may change between different institutions and are independent of the semantics. Our approach completely separates the definition of the schema matching from the actual execution. This is important because different processes might have different life-cycles; in particular the schema matching definition has a different life-cycle compared to the URI generation process. The former is subject to more sparse changes compared to the latter.

In this paper, we extend our previous work in the area [52] in many ways. First, we present more information and details on the background, the related work and the *X3ML mapping framework*. Although in our past work we focused on only one of the components (particularly the *X3ML Engine* component), in this paper we present in detail the framework as a whole and describe all its relevant components; just indicatively the key components of the framework are: (a) ***Mapping Memory Manager***, (b) ***3M Editor***, and (c) ***X3ML Engine***, however there are also additional components that are described in detail in Section 4. In addition, in this paper we demonstrate the feasibility of our solution out of various project experiences. More specifically, we describe how the proposed framework has been exploited for supporting the mapping and transformation of several archives and databases from various (even heterogeneous) domains, to CIDOC CRM [35] and its extensions [34]. Finally, we discuss the usability aspects of some of these components and demonstrate our current advances in the area.

In the sequel we use the following terminology: (a) we refer to the *X3ML mapping framework* as the X3ML framework, and (b) we refer to the *X3ML mapping definition language* simply as X3ML.

The remainder of this paper is organized as follows: Section 2 discusses the related work, whereas Section 3 presents the background for understanding the context of the X3ML. Section 4 presents the overall architecture of the X3ML framework providing more details on the individual components. Section 5 presents experiences on using the X3ML framework demonstrating the feasibility of our solution and the great advantages gained. Finally, Section 6 concludes this paper and discusses the future directions of our work.

2 Related Work

In the latest years an active field of research is focused on mapping relational databases (RDB) to RDF, since the majority of data currently published on the web are still stored in relational databases with local schemata and local identifiers. Bridging the conceptual gap between the relational model and RDF is the key to make the data available as linked data, materializing the vision of Semantic Web [24].

One approach towards this direction is the Direct Mapping [23,41] which maps automatically relational tables to classes, and attributes of tables to properties using an RDF vocabulary. The URIs of the instances and the classes are automatically generated based on the RDB schema and data. One implementation that exploits Direct Mapping is SquirrelRDF [12]. This approach is based on mapping discovery, and URI generation is tied to the implementation of the system. This limits the ways to generate and use URIs, making therefore difficult the support of complex structures and information integration.

Besides Direct Mapping, there are also XML-based solutions such as D2R MAP [26] which is a declarative language for describing mappings between relational databases and OWL/RDFS ontologies. D2R MAP is extended by eD2R [21] to map databases that are not in first normal form and by D2RQ [27], which is formally defined by an RDFS schema. The mappings are based on SQL queries that extract records from the RDB and transformation functions that can be applied to the extracted values.

Another XML-based approach, focusing on expressivity, is R2O [22] which is able to cope with complex mapping cases where one model is richer than the other. For that sake, Virtuoso RDF Views [7] are created, exposing RDBs, using a declarative meta schema language for defining the mapping of SQL data to RDF vocabularies. Triplify [20], on the other hand, maps HTTP-URI requests onto RDB queries and translates the resulting relations into RDF statements.

R2RML [31] is a mapping language proposed by W3C in order to standardize RDB to RDF mappings. There exist already several implementations [9] and R2RML is lately extended by RML [33] to simultaneously support different mapping sources into RDF. Similar to the previous solutions is the Relational.OWL [32] which is a representation format for mapping relational models to OWL full ontologies.

Besides approaches that try to map relational data, there are other approaches which try to map CSV files to RDF. For example XLWrap's mapping language [49] provides conversions from CSV and spreadsheets to RDF data model. Mapping Master's M2 [53] converts data from spreadsheets into OWL statements and Vertere [15] is a conversion tool based on a templating mechanism. Each line results in one or more RDF resources, while each column value can result in one or more triples about this resource. TaRQL [13] is used also for the same purpose. Other tools that provide mappings from XML to RDF lead to mappings in the syntactic level rather on the semantic level and fail to provide solutions applicable to broader domains. There are tools in this category that are based on XSLT (Krextor [48], AstroGrid-D [2]), on XPATH (Tripliser [14]) and XQUERY (XSPARQL [25]) and tools that are based on algorithms (GRDDL [30]) that provide links between data and RDF. Finally, there are existing tools that provide mappings from several formats to RDF. Tools in this category include Datalift [55], The DataTank [3], OpenRefine [8], RDFizers [10], Virtuoso Sponger [16]. A fundamental problem when using highly expressive languages such as XSLT is the fact, that even the programmer himself has difficulties to understand the mapping logic. The only way to verify the mapping is testing its output behaviour.

Other approaches exploit mapping technologies to publish their data as linked data. For example the Smithsonian American Art Museum[1] used KARMA [58] to publish their data as linked data, a tool trying to automate the mapping process and allowing the users to adjust the generated mappings. However, there is still no clear distinction on the work of the domain and the IT experts, which perplexes the whole workflow. KARMA uses R2RML model inheriting the issue of tight coupling between the schema matching and the URI generation.

One work with similar goals to our approach is the SIP Creator [11], created for Europeana[2] in 2009 to bring together more than 150 different sources. Whereas it only dealt with flat formats at the beginning, it was later expanded to handle hierarchical formats as output, however only in XML. Motivated by the goal of transparency, the mapping file format created by the SIP Creator was seen to come closer to the structure that was required for generic mapping, since it appeared in an easy-to-comprehend, human readable XML format with the source and target paths clearly identified. However it combined interactive schema matching resolving not formally structured elements (they call it "syntax normalization"), on the basis of the Groovy [4] programming language. Groovy however, as a programming language cannot be used or understood by a domain expert.

Most of the described approaches tightly couple the URI generation and the schema matching processes and lack general conditions where mapping is dependent on particular data values, or data structures, limiting the ways that the URIs can be constructed and making it impossible to select other forms of URIs to be generated. The latter may frequently be required in order to adapt transformed data to different targets, such as Linked Open Data (LOD), or to look-up the identifiers used in the target system. Furthermore, in the described works there is no distinction between IT and domain experts, so the IT experts are assumed to be responsible for the entire process. This is reflected in the complexity and lack of user-friendliness of the employed mapping languages, even if some easy parts of the mapping are done graphically. Furthermore, IT experts cannot easily understand the domain semantics they are dealing with and URI generation is still based on a sort of unique key generation from data, as in Relational Database Management Systems (RDBMS).

All these different approaches prove that there is no standard model to support mapping of data sources other than relational, the technologies used are too complex to be used by the domain experts and the whole workflow is not well-defined. Compared to these approaches our work:

(a) uses a simple model for defining the mappings in a way that is comprehensible and readable by the domain experts,
(b) is generic because the mapping definitions are not tied to the implementation of the data transformation engine,
(c) supports incremental changes of source and target schema,
(d) supports customized URI generation policies, and
(e) promotes the collaborative work of experts with different roles on the mapping process.

[1] `http://americanart.si.edu/`
[2] `http://www.europeana.eu/`

	Model/ Language/ Specification	Software Platform/ Tool
Relational-based	Direct Mapping [23] R2RML [31] RML [33] Relational.OWL [32]	Squirrel RDF [12] RDB2RDF [9] Virtuoso RDF Views [7] Triplify [20] D2RQ [27]
XML-based	D2R Map [26] eD2R [21] R2O [22] GRDDL [30]	Krextor [48] AstroGrid-D [2] Tripliser [14] XSPARQL [25]
CSV-based	XLWrap [49]	Master's M2 [53] Vertere [15] TaRQL [13]
multiple		DataLift (DB, CSV, XML, GML, etc.) [55] The DataTank (CSV, XLS, XML, JSON, etc.) [3] OpenRefine (CSV, XML, JSON, RDF. etc.) [8] RDFizer (many) [10] Virtuoso Sponger (many) [16] KARMA (DB, CSV, JSON, XML, XLS) [58]

Table 1 A categorization of the related works

Table 1 shows the related approaches, categorized with respect to the different source types.

3 Background

Our work has been based on two main pillars: (a) the Synergy Reference Model, that describes the best practices for the data provisioning and aggregation processes and (b) the *X3ML mapping definition language* (or simply X3ML), for describing the mappings per se. Below we will describe them in detail.

3.1 Synergy Reference Model

The Synergy Reference Model (for short SRM) which is an initiative of the CIDOC CRM Special Interest Group[3], is a reference model for a better practice of data provisioning and aggregation processes, primarily in the cultural heritage sector, but also for e-science. It is based on experience and evaluation of national and international information integration projects. It defines a consistent set of business processes, user roles, generic software components and open interfaces that form a harmonious whole. Currently a draft version of the model is available online[4], still being evolved and enriched. The goal of SRM is to: (a) describe the provision of data between providers and aggregators including associated data mapping components, (b) address the lack of functionality in current models (i.e., OAIS [50]), (c) incorporate the necessary knowledge and input needed from providers to create quality sustainable aggregations and, (d) define a modular architecture that can be developed and optimized by different developers with minimal inter-dependencies and without hindering integrated UI development for the different user roles involved.

SRM aims at identifying, supporting or managing the processes needed to be executed or maintained between a provider (the source) and an aggregator (the target) institution. It supports the management of data between source and target models and the delivery of transformed data at defined times, including updates. This includes a mapping definition, i.e., specification of the parameters for the data transformation process, such that complete sets of data records can automatically be transformed. A high level view of the data provisioning process is shown in Fig. 1.

In more details, the main steps of the data provisioning workflow are:

– **Schema matching**: source and target schema experts (a.k.a the domain experts) define a schema matching which is documented in a schema matching definition file. This file should be human and machine readable and it is the ultimate communication means on the semantic correctness of the mapping.
– **Instance generation specification**: in this step the URI generation and datatype conversion policies are defined for each instance of a target schema class referred to in the matching. In this step only IT experts are involved and domain experts have no interest or knowledge about it.
– **Terminology mapping**: the terminology mappings between source and target data/terms are defined. Providers may use anything from intuitive

[3] `http://www.cidoc-crm.org/who_we_are.html`
[4] `http://www.cidoc-crm.org/docs/SRM_v1.4.pdf`

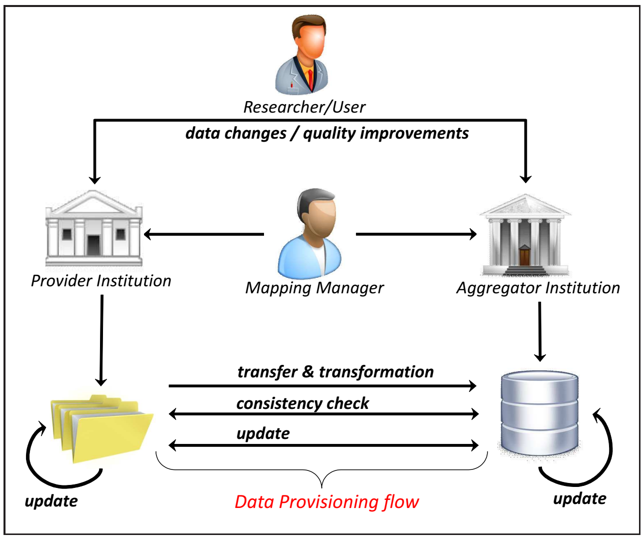


Fig. 1 The data provisioning process

lists of uncontrolled terms up to highly structured third party thesauri.

- **Transformation**: once the mapping definition has been finalized (and all syntax errors are resolved) the data needs to be transformed, producing a set of valid target records. The transformation process itself may run completely automatically. In the case where any issues arise, the aggregator can resolve them on a temporary or permanent basis but it is also possible that these records are sent back to the provider for further analysis and resolution.
- **Ingestion**: once records are transformed, an automated translation for source terms using a terminology map follows. The transformed records will then be ingested into the target system.
- **Change detection**: after the ingestion of the records all changes that may affect the consistency of provider and aggregator data are monitored. SRM foresees a series of distinct update processes in all partner sites and is the only framework at the moment which takes the maintenance into account.

3.2 X3ML Mapping Definition Language

X3ML is an XML based language which describes schema mappings in such a way that they can be collaboratively created and discussed by experts. X3ML was designed on the basis of work that started in FORTH in 2006 [42] and emphasizes on establishing a standardized mapping description which lends itself to collaboration and the building of a mapping memory to accumulate knowledge and experience. It was adapted primarily to be more aligned with the DRY principle [59] (avoiding repetition) and to be more explicit in its contract with the URI generation process. X3ML separates schema mapping from the concern of generating proper URIs so that different expertise can be applied to these two very different responsibilities.

Schema matching is performed by domain experts who need to be concerned only with the correct interpretation of the source schema. The structure of X3ML is quite easy to understand consisting of: (a) *a header* that contains basic information (e.g. title, description, contact persons, the source and target schemata, sample records etc.), and (b) a series of *mappings* each containing a domain (the main entity that is being mapped) and a number of links which consist of a path and a range. Each link describes the relation (path) of the domain entity to the corresponding range entity.

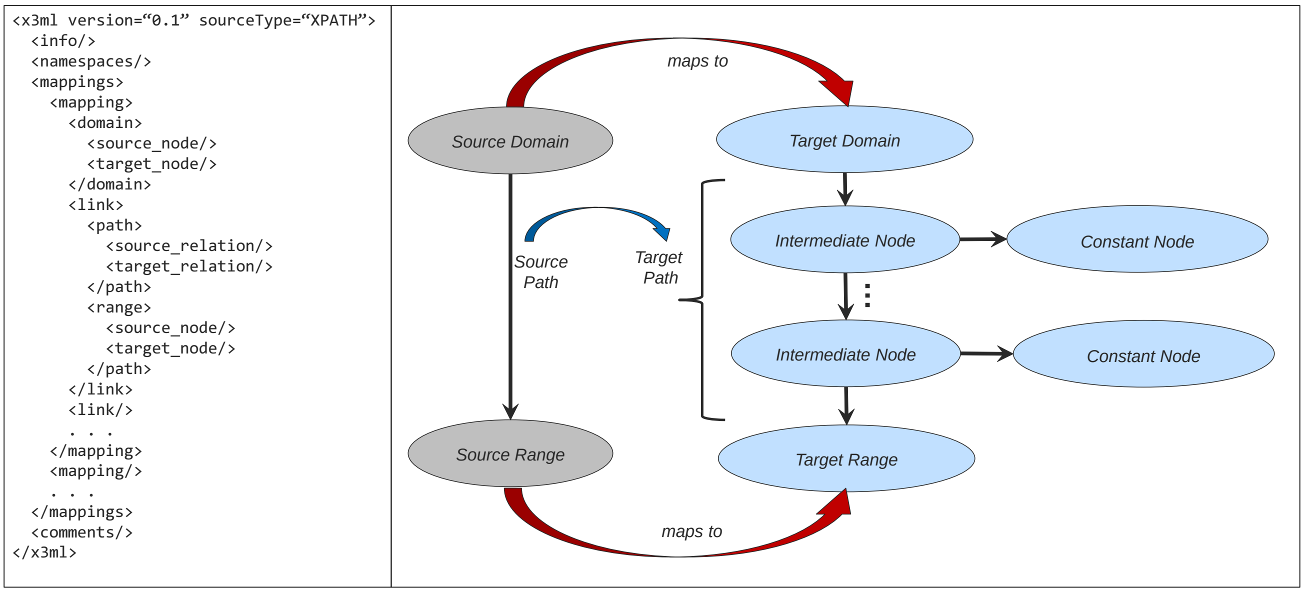
The basic mapping scheme and the XML representation of an X3ML mapping is shown in Fig. 2. Each *domain-path-range* of the source schema is mapped individually to the target schema and can be seen as self-explanatory, context independent proposition. As it is evident from the figure, X3ML mappings allow the definition of rich structures by adding intermediate nodes, even if the source relations are much simpler. An X3ML structure consists of:

- the mapping between the source domain and the target domain
- the mapping between the source range and the target range
- the proper source path
- the proper target path
- the mapping between source path and target path.

Below we will describe the main concepts of the X3ML mapping definition language.

***Info* and *Comment*:** since X3ML is intended to bridge the gap between human authors and machines, it has to allow textual comments to be placed in the mapping specification. This is intended for describing alternatives to be discussed between experts and for harmonizing parallel mappings between providers. For this purpose the info and the comment elements have been defined in the mapping specification. These blocks contain relevant information for humans to understand the specific mappings and can also contain useful provenance information recorded automatically by the tools used to manage the X3ML files, such as the date of creation or the author of the mapping file.

***Mapping*:** each mapping element consists of a domain element and a number of links. It is quite common to have a single domain mapping and some range mappings, so by using this particular format for mappings, the single domain mapping doesn't have to be declared again. This is an ergonomic choice good for



```
<x3ml version="0.1" sourceType="XPATH">
  <info/>
  <namespaces/>
  <mappings>
    <mapping>
      <domain>
        <source_node/>
        <target_node/>
      </domain>
      <link>
        <path>
          <source_relation/>
          <target_relation/>
        </path>
        <range>
          <source_node/>
          <target_node/>
        </path>
      </link>
      <link/>
      . . .
    </mapping>
    <mapping/>
    . . .
  </mappings>
  <comments/>
</x3ml>
```

Fig. 2 The structure and the XML representation of an X3ML mapping

tree-dominated source schemata or sets of relational tables, which helps user orientation. It further provides an intuitive default local scope to define that the same instance of a class in a mapping rule appears as domain value in multiple target propositions.

***Domain*:** the domain element is used to specify the mappings between a source (`source_node`) entity (table, class, non-leaf element) that can be regarded as domain of a source proposition and an equivalent target (`target_node`) entity. The `source_node` provides information on how to navigate to the source record and in case of XML it is an XPATH expression. The `target_node` defines an entity element that will lead to the generation of resource URIs or datatype values for the output graph. It may also contain `if` conditions (described below) upon which the mapping depends on.

***Link*:** inside the link element there is a path element. It allows mapping a source relation from the above defined source domain to a target relation to the above defined target domain. The path element must be followed by a range element, which is used to map the source and the target entities that are the equivalent range of the respective paths. The target relation might contain `if` conditions as well. A source / target range pair may reappear as a subsequent domain in an X3ML mapping.

***Conditional*:** the conditional expressions in *X3ML mapping definition language* can check for existence, equality and narrowness of values. They are expressed in the form of `if` statements and they can be combined into boolean expressions.

***Intermediate node*:** sometimes a path in the source schema needs to be further analyzed to a sequence of paths in the output with respect to the target schema. For this reason the user can define the generation of an intermediate node (or intermediate entity).

***Additional*:** regularly constant properties and entities are needed to be added to a target entity, either from background knowledge or in order to characterize the meaning of a classification by the source schema rather than by data. For instance, a database about museum objects may not mention at all the museum as current keeper. A table "coin" may be mapped to "physical object", but each instance of "coin" must have the type "coin". For that purpose, an additional element can be used, containing the entity which will be attached to the target entity, the relationship describing the link, and the respective constant values.

***Variables*:** sometimes it is necessary to generate an instance in X3ML only once in the scope of a given domain entity, and then re-use it in a number of links of this domain. This is most frequently the case for intermediate target nodes. For example, a description of a museum object may re-use the same production event for mapping its "creator" link and its "date" link. In these cases, an entity can be assigned to a variable.

***Join* operator:** sometimes it is required to combine values from different tables in the source and produce new values in the output. This is the definition of the relational join operation. X3ML contains a specific operator for support (n-ary) join operation between different tables. The operator that is used is `'=='` and is being used inside a link element. More specifically it is being expressed inside the path element and expresses the equality of the value of the left-hand side (its table is the one defined the in the domain of the corresponding mapping) with the value of the right-hand side (its

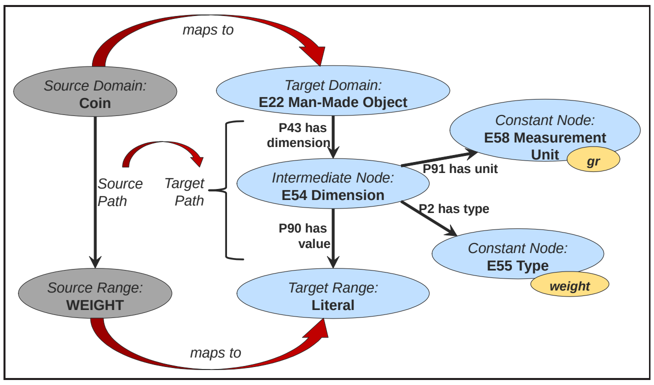


Fig. 3 Mapping relational data to CIDOC CRM

table is the one defined on the range of the corresponding link).

Instance generation policy: the definition of the URI generation policy follows the schema matching and is performed usually by an IT expert who must ensure that the generated URIs match certain criteria such as consistency and uniqueness. A set of predefined URI generators (UUIDs, literals) and templates are available but any URI generating function can be implemented and incorporated in the system. In the X3ML definition, the target domain and range contain the functions that generate URIs or literals.

Fig. 3 shows how a simple relational database entry that specifies the weight of a coin is mapped and expressed with respect to the CIDOC CRM schema [35]. The mapping of this example can be found online[5].

4 The X3ML Mapping Framework

The X3ML framework consists of a set of software components that assist the data provisioning process for information integration. A diagram presenting these components and the flow of data among them is shown in Fig. 4.

The process starts with the Provider Institution and is completed at the Aggregator Institution, when all records are transformed to the target format and are ingested to the target system.

Starting from the Provider Institution, the ***Syntax Normalizer*** can be used to normalize the provider's records. It exploits local syntax rules and produces a new provider schema definition, called *Effective Provider Schema*. Normalization is quite often needed in date fields or in fields that contain concatenated information. For example, in the source schema the field:

```
<Acquisition>
  bequeathed; 1936-07-07; Young, Arthur W.
</Acquisition>
```

contains information about the actor, the date and the type of an acquisition and needs to be normalized before it is actually mapped to the target schema. In this example the *Effective Provider Schema* will be:

```
<Acquisition>
  <AcqType>bequeathed</AcqType>
  <AcqDate>1936-07-07</AcqDate>
  <AcqActor>Young, Arthur W.</AcqActor>
</Acquisition>
```

The next step of the provisioning process is the definition of the mappings. The X3ML framework provides various components that assist the experts to complete this time consuming and error prone task. One of these is the ***Mapping Memory Manager*** (for short ***3M***). 3M is a managing system suitable for handling the mapping files. It offers a variety of actions that help both provider and target schema experts manage their files and also communicates with another component, called ***3M Editor***. 3M Editor is an application suite that helps the experts complete the mapping definition process. The first step of the mapping definition is the schema matching. Provider schema experts with target schema experts, exploit the ***Schema Matcher*** component, in order to define a schema matching which is documented in a *Schema Matching Definition*. The ***Source Schema Validator*** and the ***Target Schema Validator*** components assist the experts in selecting the valid paths with respect to the corresponding schemata, preventing them from making mistakes, while the ***Source Schema Visualizer*** and the ***Target Schema Visualizer*** assist users navigating through all source and target elements. The matching process is also supported by the ***Mapping Suggester***, which makes use of "mapping memories" of similar cases as they are collected and cached from the user community.

The next step is the specification of the instance generation rules that define the URI generation policy for each target class. This task is accomplished by the ***Instance Generator Rule Builder***, which complements the Schema Matching Definition with the instance generation policies, producing the *Mapping Definition* (which is also called X3ML file). Both the Schema Matching Definition and the Mapping Definition may be viewed with the ***Schema Mapping Viewer***. The files are stored to the *Mapping Memory*, which is an XML database.

[5] http://www.ics.forth.gr/isl/OEAWcoins-Published

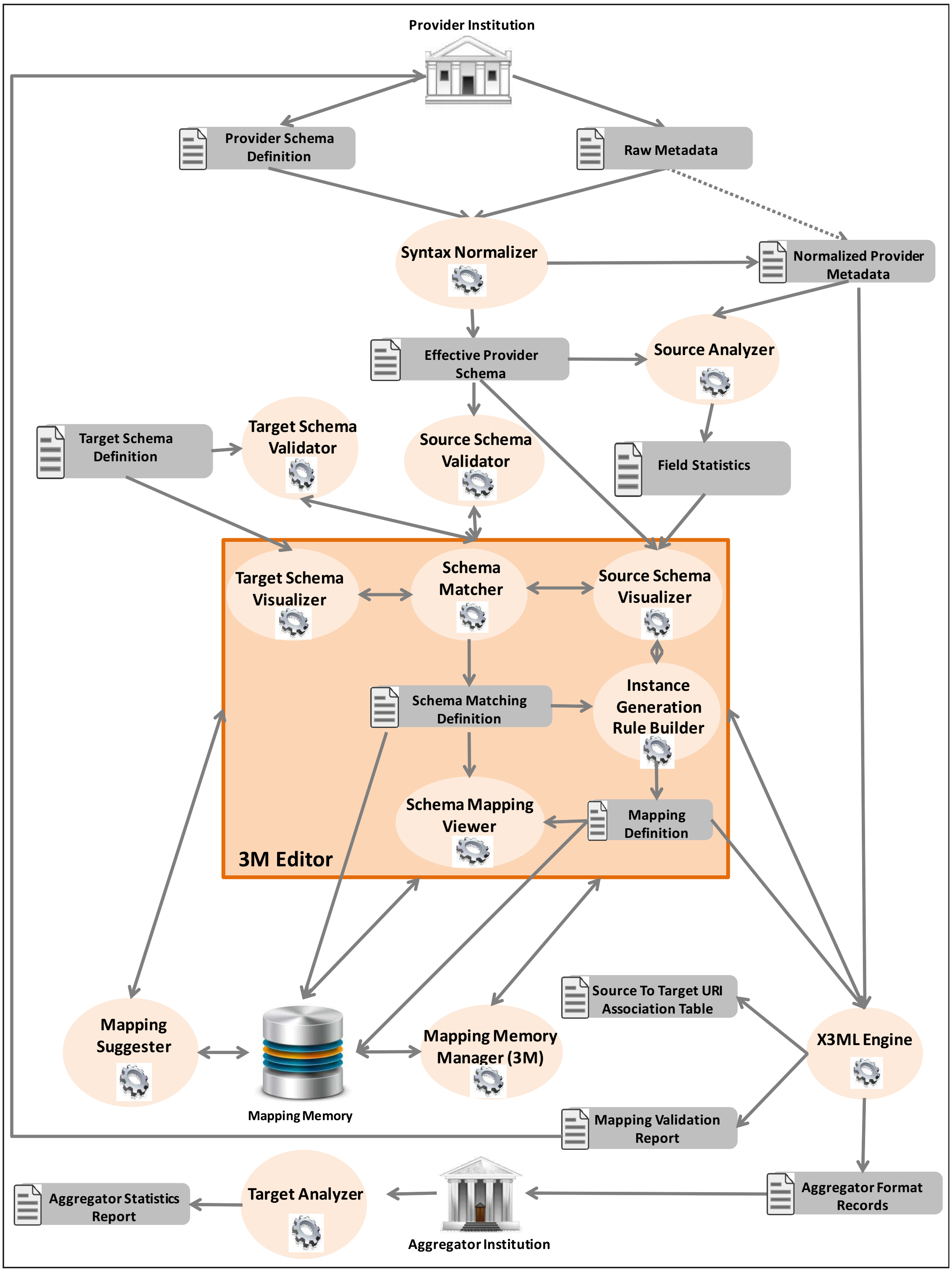


Fig. 4 The X3ML Flow Network

Afterwards, the mapping definition and the normalized input from the provider, are given as input to the ***X3ML Engine*** component which is responsible for transforming the records to the aggregator format, so that they can be ingested to the Aggregator Institution.

The components of the X3ML framework have been designed with respect to the following design principles:

– *Collaborative Mapping Memory.* The X3ML mapping descriptions must lend themselves to being stored and handled by collaborative tools, as well as potentially written by hand. This was the motivation for choosing a simple syntax in XML, and one which does not depend on implicit knowledge.
– *Facilitating Instance Matching.* This involves extracting semantic information with the intent of generating correct instance URIs.
– *Transparency.* The most important feature of X3ML framework is its generic application to mapping creation and execution and hopefully its longevity. The cleaner the core design of the framework and X3ML specication, and the clearer its documentation, the more readily it will get traction and become the basis for future mappings.
– *Re-use of Standards and Technologies.* The best way to build a new software module is to carefully choose its dependencies, and keeping them as small as possible. Building on top of proven technologies is the quickest way to a dependable result.

In the following subsections, we will describe these components in more detail.

4.1 Syntax Normalizer

The Syntax Normalizer component is responsible for converting all data structures that are necessary for the transformation into a standard form. The reason for converting them, is because data transformation components can deal with a limited set of standard data structures.

In many cases the local syntax rules of the provider can be so complicated, or even non-deterministic, that it is often more effective to use a set of custom filtering rules, that resolve one structural feature at a time, and verify it with the source schema experts. For instance, if italics are used to tag a particular kind of field, it is better to convert first all italics to XML tags, so that the semantics of the field will be properly preserved and transformed.

To this end, the Syntax Normalizer component contains the necessary filters for converting the input data structures properly. Currently three different syntax normalization filters have been implemented. The functionality and the design of the filters were based on actual requirements coming from the users of the X3ML framework.

The first filter normalizes the syntax of an XML file by rearranging its structure in order to fix syntax issues, so that it becomes compliant with the concepts of a semantic model. The second filter assists the users inspecting the values of specific elements in an XML file and update specific values while preserving the original ones. The third filter's main functionality is to split the values of an element, to create sub-elements and assign these values to them.

The following example shows an element *events* containing various dates, actors, and places:

```
<events>
  <date>2014-11-23</date>
  <date>2015-01-15</date>
  <actor>James</actor>
  <actor>Bob</actor>
  <place>Greece</place>
  <place>Italy</place>
</events>
```

In the above example clearly the sequence of the elements is important; practically there are two different events with the corresponding date, actor and place. Therefore, the Syntax Normalizer component reconstructs this block and produces the normalized one, which is shown below:

```
<events>
  <event>
    <date>2014-11-23</date>
    <actor>James</actor>
    <place>Greece</place>
  </event>
  <event>
    <date>2015-01-15</date>
    <actor>Bob</actor>
    <place>Italy</place>
  </event>
</events>
```

4.2 Mapping Memory Manager (3M)

3M is a tool for managing mapping definition files. It provides a number of administrative actions that assist the experts to manage their mapping definition files. It is available online[6] and is free to use. Users are requested to register online in order to obtain a username

[6] http://www.ics.forth.gr/isl/3M/

and password. Using these credentials users can login and see all the available mappings. Although all the mappings are visible to all users, each user is allowed to edit only his own mappings and the mappings he has access rights to do so. Users can also create a new mapping file, by giving a title and selecting one or more of the proposed target schemata. The newly created mapping file can be further edited with the 3M Editor (described in the subsequent section). Since the number of mappings can become quite big, several actions (such as searching, filtering and sorting) are provided. Moreover, users are able to export their mappings for offline use, import mapping files, make versions of a specific mapping file, delete it, make a copy and also give rights to other users to edit their own file.

4.3 3M Editor

The 3M Editor component is a web application suite containing several software sub-components and exploits several external services. It is available online[7] and its main functionality is to assist users during the mapping definition process, using a human-friendly user interface and a set of sub-components that either suggest or validate the user input.

The user interface of the 3M editor is tabular and resembles modern web browsers. The rationale for this design is to allow users quickly familiarize with the editor and navigate instantly through the offered options. The main task of the editor is to support the creation of a complete X3ML file and check how the actual source data are mapped to the defined target output.

To create a complete X3ML file the user has to fill in two different categories of resources:

– **Informational resources:** they contain some general information about the mapping and its creators, the source, the target schema and the corresponding namespaces. Additionally, they might contain examples about the source or target records or even a generator policy file (we discuss them in detail §4.7).
– **Mapping resources:** they contain information about the mappings themselves. Users are facilitated with the definition of the mappings using a set of supporting sub-components. A core sub-component is the Schema Matcher, which supports the definition of mappings. To define them, users are working with a tabular format of the mappings. We use this design since we found out that most users were accustomed to maintaining mappings of their own in spreadsheets. Thus each mapping is represented as a table. Fig. 5 shows an actual example. The header of the table represents the domain of the mapping, and the rows represent the links. Since each link contains two elements; one path and one range, the rows are double in size and contain both these elements. The columns of the table are used to separate the expression in the source schema (i.e., the Provider's schema), from the expression in the target schema (i.e., the Aggregator's schema), as well as conditional expressions or comments (for humans).

[7] Through the 3M component

Although the creation of an X3ML file is the main functionality performed by the 3M Editor, it provides also the following:

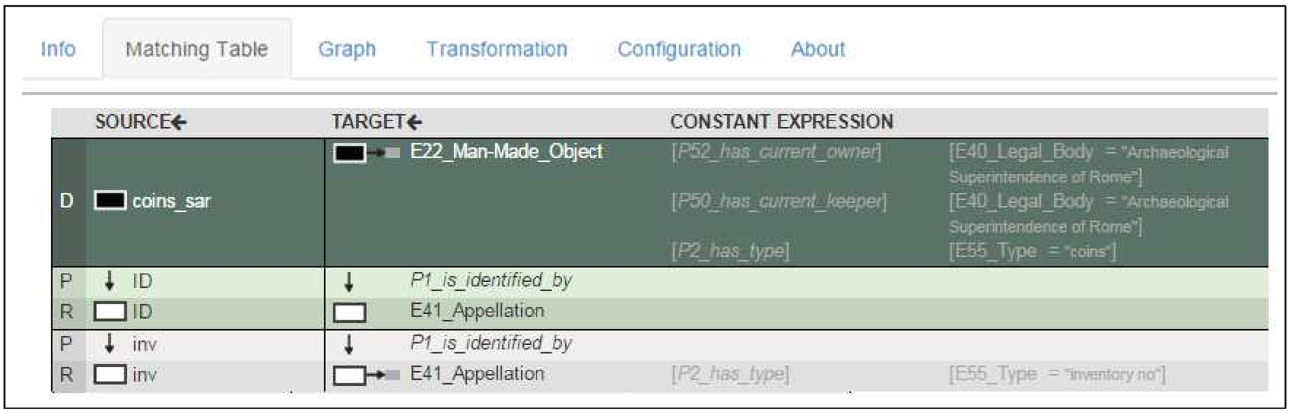
– constructs a graphical visualization of the mappings, for facilitating non-experienced users understanding the mappings, and quickly inspecting them. An indicative screenshot of the produced graph is shown in Fig. 6.
– executes the X3ML engine (described later in §4.7) and inspect how source records will be transformed with respect to the defined mappings. More specifically, users can upload their source records and a generator policy file (described below in §4.3.4) and select the target records format. This will allow transforming the source records by exploiting the X3ML engine component.
– configures various options to better suit the individual mapping needs; for example user may change view options or enable/disable sub-components.

4.3.1 Schema Matcher

The Schema Matcher is the core sub-component of the 3M Editor. It allows the user defining a schema matching expressed in X3ML language, in an intuitive and user friendly way. Users can define meaningful matching definitions, that preserve the semantics of the source records, by adding the appropriate information.

The user adds information about the mapping and declares a source schema and one or more target schemata with their namespaces. The title and the target schemata (accompanied by their namespaces) may have already been filled by 3M (described in §4.2) when creating the mapping file. In addition, the user can also upload files that are going to be used as source or target schemata, example source or target records and a generator policy file. The uploaded files are then parsed, analyzed and used by the corresponding components.

The component performs its functionality on an interactive manner; at any given moment, the user can switch from view mode (that uses the Schema Maping



Info | Matching Table | Graph | Transformation | Configuration | About

	SOURCE	TARGET	CONSTANT EXPRESSION	
D	coins_sar	E22_Man-Made_Object	[P52_has_current_owner]	[E40_Legal_Body = "Archaeological Superintendence of Rome"]
			[P50_has_current_keeper]	[E40_Legal_Body = "Archaeological Superintendence of Rome"]
			[P2_has_type]	[E55_Type = "coins"]
P	ID	P1_is_identified_by		
R	ID	E41_Appellation		
P	inv	P1_is_identified_by		
R	inv	E41_Appellation	[P2_has_type]	[E55_Type = "inventory no"]

Fig. 5 Specifying and inspecting mappings using 3M Editor

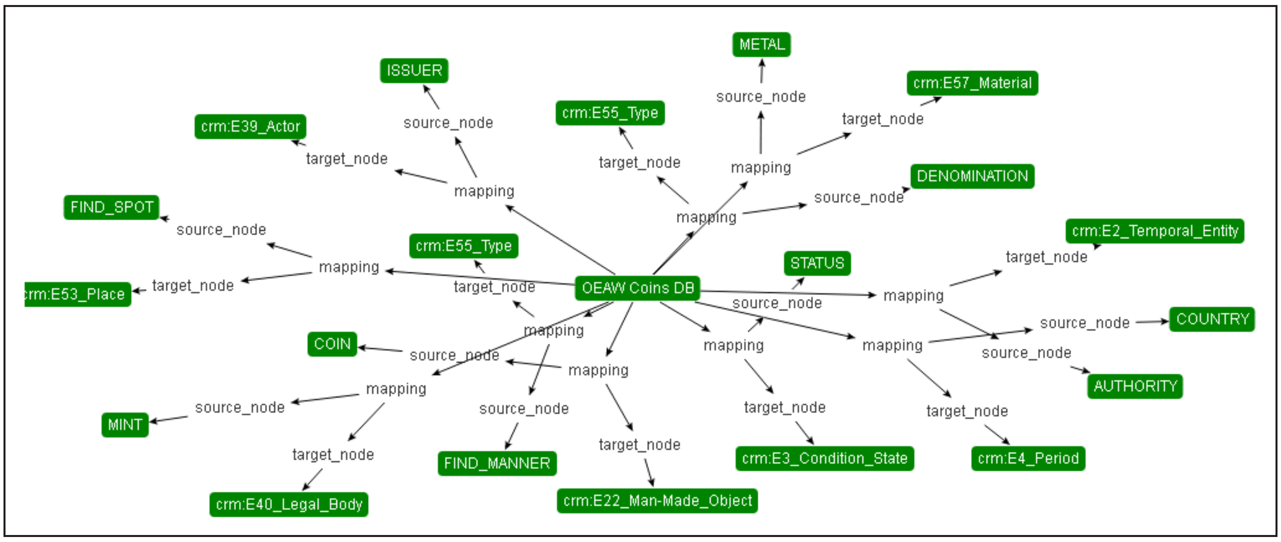
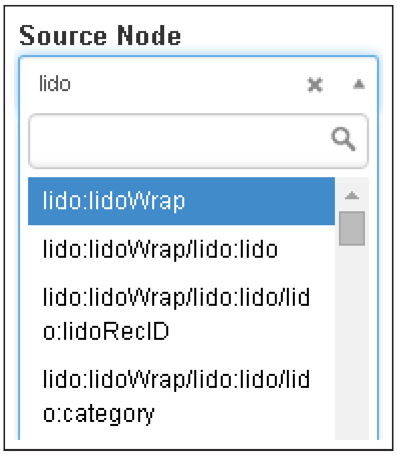


Fig. 6 3M Editor graph Tab

Viewer component - described in §4.3.5) to edit mode, for inspecting how the mapping process is progressing. Furthermore if the users are familiar with the *X3ML language* syntax, there is even a raw XML mode.

4.3.2 Source Schema Visualizer

The Source Schema Visualizer component is responsible for assisting users in selecting the appropriate source path for the definition of the mapping. This component replaces the input fields with select boxes that contain values from the source records, so that users can select the values instead of writing them explicitly. The component analyzes the structure of the source schema and "proposes" values to the user, to avoid mistakes in writing the XPATH expressions. It is evident that the source schema file, or the source records should be uploaded beforehand. The component can be activated or deactivated through the 3M Editor component. An indicative screenshot of the component is shown at Fig. 7. The source schema that is being exploited in this example is LIDO [29].



Source Node

lido

lido:lidoWrap

lido:lidoWrap/lido:lido

lido:lidoWrap/lido:lido/lido:lidoRecID

lido:lidoWrap/lido:lido/lido:category

Fig. 7 Selecting values from the source schema (LIDO) using Source Schema Visualizer

4.3.3 Target Schema Visualizer

The Target Schema Visualizer is a software component for assisting users in selecting the appropriate target paths. When there are no target schema files available, users have to fill in the target paths using text input fields. If at least one target schema file is available, users can exploit this component for selecting the appropriate concepts (i.e., classes) from the target schema for their mappings.

The visualizer is activated after a target schema has been uploaded and validated. For the latter the Target Schema Validator sub-component is being exploited (described in more detail in §4.4). Fig. 8 shows an indicative screenshot of the Target Schema Visualizer. The available options that are shown are classes of the target schema; in this case the target schema is CIDOC CRM.

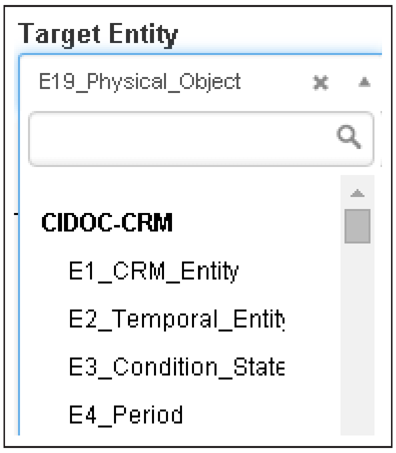


Fig. 8 Selecting values from the target schema (CIDOC CRM) using the Target Schema Visualizer

4.3.4 Instance Generation Rule Builder

After the creation of the *Schema Matching Definition*, the specification for the instances generation is defined. This is accomplished using the Instance Generation Rule Builder software component. The user interface is similar to the Schema Matcher component (described in §4.3.1). However, users can only edit details about the instance and value generators.

The component uses templates for specifying how a URI or a label will be created and they can be exploited throughout the mapping. The generator templates are defined separately and linked to the actual mapping, in order to be used by the generators for producing the actual identifier or label. Each target entity must have only one instance generator and any number of label generators. Generators use their names as identifiers, therefore the same generator can be exploited for many different target entities. There are some generators already built-in in the X3ML framework, however more user-defined generators can be defined. A more detailed discussion about generators can be found in the sequel (in §4.7).

Figure 9 shows an indicative example; the upper part defines how the URIs for instances of the class `E41_Appellation` will be assigned, while the lower part defines how the corresponding labels will be created (e.g., `rdfs:label`).

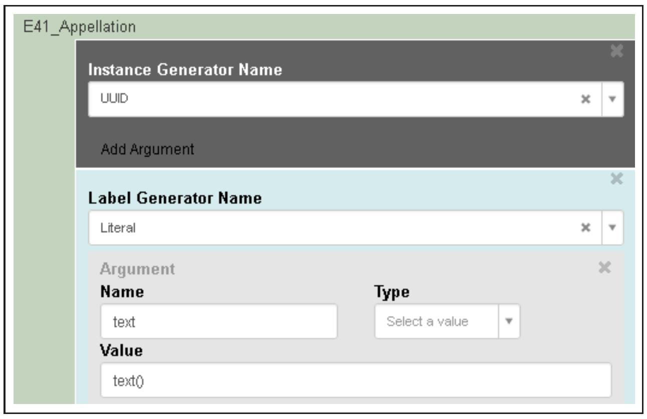


Fig. 9 Instance Generation Rule Builder

4.3.5 Schema Mapping Viewer

Schema Mapping Viewer is a sub-component used to inspect X3ML files. There are multiple interaction modes:

- **View mode**: This mode shows the mappings using a human understandable representation. More specifically, it uses a tabular format where each mapping is modeled as a different mapping, the header of the table represents the domain of the mapping, and the rows represent the links. This is the default mode for showing the mappings and is also used to present the publicly available mappings with respect to CIDOC CRM[8].
- **Raw XML mode**: X3ML is presented as raw XML. It may be the entire X3ML file or specific subtrees. This mode is intended to be used from users who are familiar with the syntax of the X3ML.
- **Graph view mode**: X3ML is presented as a graph (shown in Fig. 6). This mode is intended for users non-experienced with the X3ML, or for quickly getting an overview of the defined mappings.

[8] `http://www.ics.forth.gr/isl/3M-PublishedMappings/`

4.4 Source Schema and Target Schema Validators

The Source Schema Validator is an external tool used to validate and traverse the source schemata and produce all possible XPATH expressions. The identified expressions are given as input to the Source Schema Visualizer for "proposing" them to the user. The supported source schemata may be in XSD or XML schema format.

The Target Schema Validator is a tool used to analyze the target schemata and to produce valid options according to previous user selections. The target schemata may be in RDFS, RDF, OWL, XSD or XML format. It consists of a set of engines for validating the schema based on their format. Currently the following Target Schema Validator engines are implemented:

– using XML structure: It works with RDFS or RDF schema files. The engine is based on queries evaluated over the RDFS schemata stored in an XML database. Schemata have to be well formed XML files containing certain tags (`rdfs:Class`, `rdf:Property`, `rdfs:domain` etc.). The results are sorted and grouped by schema file.
– using semantic reasoner[9]: It works with RDFS, RDF or OWL schema files. The engine is based on Jena[10], and exploits also inference on the schema.
– Using XML analysis: It works with XSD or XML schema files. The engine produces XPATH expressions as result.

4.5 Source and Target Analyzers

The Source and Target Analyzers are two components under development that aim to provide information regarding the coverage of the source and target schemata respectively.

The Source Analyzer contains three different metrics related to the source schema. The first metric displays the percentage coverage of parent elements, while the second counts only the leaves of the source schema that exist in the mapping. Finally, the third metric calculates the total coverage of the source schema.

The Target Analyzer allows users to explore the target schema, offering an effective schema analysis. The metrics are divided into two categories:

– Direct
– Ancestors / Descendants

At the category of Direct metrics each node is a separate element. Three different rates are provided, about classes, properties and resources. An element is considered to be covered only if it is mentioned in the mapping directly. On the other hand, the category of Ancestors / Descendants provides the same rates (classes, properties and resources), but it is differentiated on how the covered elements are calculated. An element is considered to be covered, if at least one ancestor or the element itself is referred in the mapping.

4.6 Mapping Suggester

The Mapping Suggester [57] is a software component under development used to suggest mappings to the user. These suggestions make use of "mapping memories" of similar cases collected from the user community and are recalculated with each new mapping decision. The user can either accept or reject the suggestion. When the user creates a new mapping file, the Mapping Suggester runs a schema matching with the source schema they provide and the existing mapped source schemata in the mapping memory. The correspondences/crosswalks found during the schema matching, are used by the mapping suggester in order to suggest mappings to the user.

4.7 The X3ML Engine

The X3ML Engine realizes the transformation of the source records to the target format. The engine takes as input the source data (currently in the form of an XML document), the description of the mappings in the X3ML mapping definition file and the URI generation policy file and is responsible for transforming the source document into a valid RDF document which corresponds to the input XML file, with respect to the given mappings and policy.

Fig. 10 depicts the main subcomponents of the X3ML engine. The *Input Reader* component is responsible for reading the input data (currently we support XML documents, however as we describe in Section 6 more formats will be supported in the future using the proper extensions). The *X3ML Parser* component is responsible for reading and manipulating the X3ML mapping definitions. The component *RDF Writer* outputs the transformed data into RDF format. The *Instance Generator* component produces the URIs and the labels based on the descriptions that exist in the mappings and finally the *Controller* component coordinates the entire process.

One of the most important tasks of the X3ML engine is the generation of values. Values can be either literal values or identifiers for resources (i.e., URIs). The

[9] `https://github.com/isl/Reasoner`
[10] `https://jena.apache.org/`

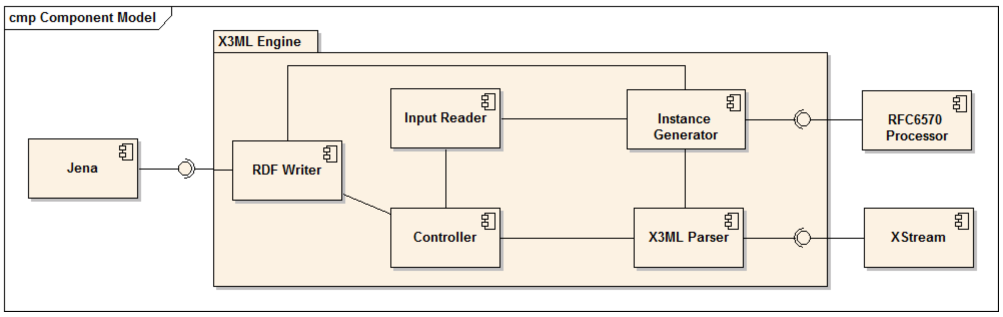


Fig. 10 The main components of X3ML Engine

generation of values is being handled by the *Instance-Generator* component. In general this component supports the generation of: (a) instances and (b) labels. For each entity there must exist one `instance_generator` and any number of subsequent `label_generator` blocks in the corresponding mapping definition file.

The following block shows a template for defining an `instance_generator`.

```
<instance_generator name="[gen-name]">
   <arg name="[arg-name]" type="[arg-type]">
      [arg-value]
   </arg>
   ...
</instance_generator>
```

This type of generator is responsible for constructing an identifier for a resource. Each `instance_generator` should be assigned a name, and associated with a list of arguments. The arguments are being exploited to provide the text segments that are required for constructing an identifier. Each argument has a name, which should be unique in the context of the generator, and a type and value pair. The type determines how the value of the argument will be used; there are 3 different options for the type:

1. `constant`. In this case the value of the argument should be used as it is defined. This type is used when we want to assign constant values to the generated identifiers.
2. `xpath`. In this case the value of the argument is an XPATH expression, that should be evaluated with the given input. The result of the XPATH expression will be used for the generation of the identifier. This type is used when it is required to exploit data from the input file in the generated identifiers.
3. `position`. In this case the value of the argument is ignored and the index position of the corresponding source node, within its context, is being used.

The X3ML engine provides the default implementation for producing URIs and UUIDs.

The following block shows the configuration of a `label_generator`.

```
<label_generator name="[gen-name]">
   <arg name="[arg-name]" type="[arg-type]">
      [arg-value]
   </arg>
   <arg name="language" type="constant">
      [language-code]
   </arg>
   ...
</label_generator>
```

Each `label_generator` has a name and a list of arguments (that are similar in spirit with the arguments that are defined for the `instance_generator`). One extra argument that is being exploited is the argument for defining the language tag of the generated value. If it is empty then it is implied that the generated value will not have it (i.e., in the case of number values). The X3ML engine provides default implementations for producing literal values in the form of `rdfs:label` and `skos:prefLabel` and constant values.

The *InstanceGenerator* component is configured through an XML file (which is given as input in the X3ML engine). When URIs are to be generated on the basis of source record content, it is wise to leverage existing standards and re-use the associated implementations. For the template-based URI generation, the RFC 6570 [40] standard is available and is used. Whenever the required URIs or labels cannot be generated by the default generators, the simple templates, or the URI templates, it is always possible to insert a special generator in the form of a class implementing the *Instance-Generator* component interfaces.

The X3ML engine keeps an association table for the produced "values". This practically means that whenever a new "value" is being created (either a URI, UUID, or literal) the exact XPATH from the source file is associated with the generated value. The contents of this association table can also be exported in XML format, and can be exploited from the user as a quick overview of the generated values for particular resources of the source.

4.8 Implementation Details and Software Licenses

The X3ML framework has been implemented in JAVA respecting the principles that were described in §3.2. The Web application components of the framework have been designed and implemented following a three-tier architecture [37]. More specifically it comprises the following layers:

– **Data tier**: data is stored in an XML database. For the implementation of the Mapping Memory we have used eXist DB[11].
– **Logic tier**: server code is implemented using J2EE. Whenever it was required we re-used existing and widely-adopted libraries. An indicative list contains: XStream[12] and jOOX[13] for parsing XML-based documents, and Handy URI Templates[14] to support the generation of valid URIs.
– **Presentation tier**: client code is implemented using web standards such as HTML5[15], CSS3[16], Javascript and well-known libraries and frameworks such as jQuery[17], AngularJS[18], and Bootstrap[19].

All the components of the X3ML framework have been developed as open source components. However, since they have been developed in the context of different EU projects, they are published under different open source licenses. More specifically, the 3M and 3M Editor components have been released under the European Union Public License[20] whereas the X3ML engine has been released under the Apache 2 license[21].

Furthermore, the source code of the X3ML framework has been deposited in a free and open repository. The source code and the complete documentation of the components are available from the following links:

– https://github.com/isl/Mapping-Memory-Manager
– https://github.com/isl/3MEditor
– https://github.com/isl/x3ml

11 http://http://exist-db.org/exist/apps/homepage/index.html
12 http://x-stream.github.io/
13 https://github.com/jOOQ/jOOX
14 https://github.com/damnhandy/Handy-URI-Templates
15 http://www.w3.org/TR/html5/
16 http://www.w3schools.com/css/css3_intro.asp
17 https://jquery.com/
18 https://angularjs.org/
19 http://getbootstrap.com/
20 https://joinup.ec.europa.eu/community/eupl/og_page/eupl
21 http://www.apache.org/licenses/LICENSE-2.0

5 Exploitation and Evaluation

The X3ML framework has been validated, evaluated and widely used by several projects. Below we first describe our experiences from using the X3ML framework in several EU projects (in §5.1), and then discuss about the evaluation results (in §5.2).

5.1 X3ML exploitation in EU Projects

The ResearchSpace project[22] - a collaboration of the British Museum, the Rijkmuseum, Oxford e-Research Centre, Yale Center for British Art (YCBA) and others- is developing a collaborative environment for humanities and cultural heritage research. The project has been using the X3ML framework for the mapping and transformation of their data using CIDOC-CRM as the target schema. A significant part of the British Museum and Rijksmuseum collection data have been transformed to CRM and integrated in ResearchSpace, showing the great value of the X3ML engine for transforming big amounts of real, heterogeneous cultural data.

Furthermore, the ResearchSpace project organized the CIDOC CRM Mapping workshop for humanities scholars and cultural heritage professionals, first at Yale University (USA) in August 2015, and then at Oxford University (UK) in October 2015. The workshop aimed to help professionals develop skills and understand data mapping techniques using CIDOC CRM. The participation exceeded the available positions (30 participants in Yale and 20 in Oxford), showing great interest in our approach and the potential of high impact of our work. Just indicatively the workshops attended well-known institutions in the area of cultural heritage domain, like GETTY museum[23], Canadian Heritage[24], Yale[25], Frick[26] and others. The feedback we received from the two workshops was very positive[27] and it is very promising to see that, after attending the workshops, several institutions started mapping their data, an attempt continuing to date.

Another project using and exploiting our framework is the ARIADNE project [1]. The project initiated several mapping activities using the X3ML framework, to convert existing schemata of archaeological data to

22 http://www.researchspace.org/
23 http://www.getty.edu/museum/
24 http://www.pch.gc.ca/eng/1266037002102
25 http://www.yale.edu/
26 http://www.frick.org/
27 Some feedback from the USA workshop as published in the call of the UK workshop can be found at http://www.researchspace.org/home/project-updates/cidoccrmmappingworkshopatoxforduniversity

CIDOC-CRM and its extension suite. Using the X3ML framework, ARIADNE intended to promote an open sharing of data in the archaeology sector, supporting effectively and efficiently all involved tasks. One of the tasks carried out in the context of ARIADNE project was to develop an integrated scenario where several roman coin databases are being mapped and transformed to the common CIDOC CRM schema through X3ML and finally integrated in a semantic repository. So far, five archives have been integrated:

1. a set of 72 numismatic records from the dFMRÖ archive[18], an online MySQL database of the Numismatic Research Group of the Austrian Academy of Sciences,
2. a set of 1670 numismatic records coming from the Cambridge Fitzwilliam Museum archive[28],
3. a set of 630 records coming from the Archaeological Superintendence of Rome[29] database,
4. a set of 517 coins of the Pergamon project from the Arachne object database[30] of the German Archaeological Institute,
5. the collections of MuseiDItalia, the digital library integrated in CulturaItalia with 25000 records already in CIDOC-CRM form.

The details of this integration use case are described in detail in [38]. This use case played an important role during the development of the X3ML framework both as a testbed and as a requirements provider. Furthermore, Vast-Lab[31] have been using the framework for mapping the Italian archaeological documentation system to CIDOC CRM [39].

The PARTHENOS project[32] aims at strengthening the cohesion of research in the broad sector of Linguistic Studies, Humanities, Cultural Heritage, History, Archaeology and related fields through a thematic cluster of European Research Infrastructures, integrating initiatives, e-infrastructures and other world-class infrastructures, and building bridges between different, though tightly interrelated, fields. PARTHENOS is committed to CRM-based integration and decided to use X3ML framework in order to support mappings, proving again its wide applicability and value. Towards this direction, X3ML framework has been integrated in the gCube infrastructure [56], and is publicly available[33].

In addition, X3ML has been used for mappings of the Europeana Data Model[36] and LIDO[29] to CIDOC CRM whereas VRE4EIC [17] will use the X3ML framework in order to support mappings to CERIF [19]. Moreover, the ITN-DCH project [5] has also initiated mapping activities using X3ML such as the mapping of the MayaArch3D database[34].

X3ML has also been exploited in the biodiversity domain, in the context of the Lifewatch Greece project [6]. More specifically, it has been used for describing the mappings of data from the biodiversity domain that were derived from various relational databases, as well as from CSV files, and were transformed into instances of the CIDOC CRM and its extensions for the biodiversity domain [60,61]. The aggregated records were then used for for supporting the discovery of datasets and the scientific research processes related to biodiversity domain.

All these diverse cases show an ever increasing interest in the X3ML framework and prove the effectiveness of our approach in real world cases. Figure 11 illustrates the different domains (rectangles) and the various projects (circles) where X3ML framework has been exploited.

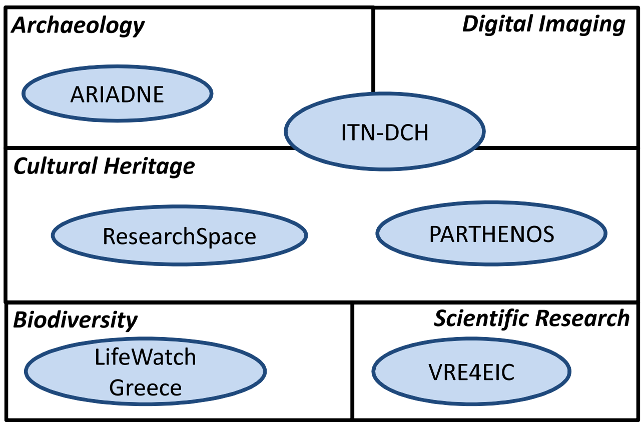


Fig. 11 The current exploitation of X3ML framework from various projects

5.2 Performance Evaluation

Apart from the evaluation over real institutions (serving real needs), we wanted to evaluate how efficient is the framework as the input grows. Therefore, we decided to evaluate the performance of the X3ML engine, since it is the component that requires the most computational resources. For the evaluation[35] we used an XML

[28] `http://www.fitzmuseum.cam.ac.uk`
[29] `http://archeoroma.beniculturali.it/en`
[30] `http://arachne.uni-koeln.de/drupal/`
[31] `http://vast-lab.org`
[32] `http://www.parthenos-project.eu/`
[33] `http://mapping-d-parthenos.d4science.org/3M/`
[34] `http://www.mayaarch3d.org`
[35] The experiments were carried out on a PC with an Intel i7 processor, 8GB RAM, running Windows 7 32 bit.

input and a X3ML mapping example coming from the ARIADNE project as a base to produce synthetic data that was provided as input to the X3ML engine. Three X3ML mapping files were created containing 10, 100 and 1000 mappings and 4 XML input files containing 10, 100, 1000 and 10000 records respectively. Fig. 12 depicts the evaluation results. We can observe that the overall time depends on both the number of mappings and the size of the input. More specifically, the time required for data transformation is approximately one second when the size of the input is low (10 records), even if the mappings are many (from 10 to 1000). As the size of the input increases however, the overall time that is required increases as well. Note, that the total number of output records is the total number of input records multiplied by the number of mappings (i.e., 10 input records with 10 mappings will produce 100 output records). Concluding, we can see that the execution time is affected equally by the number of the mappings and the records, and it is related with the number of the links that are created during the transformation process.

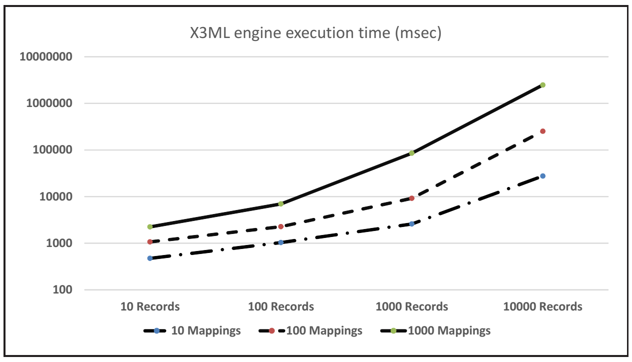


Fig. 12 X3ML Engine Evaluation Results

6 Conclusions and Future Work

6.1 Discussion

This paper presented a novel framework for the management of the core processes needed to create, maintain and manage mapping relationships between different data sources. We described the X3ML framework and its building blocks for describing both schema mappings and URI generation policies, as well as tools for managing, editing, visualizing and executing those mappings. Our framework supports the mapping definition and transformation process and the generation of URIs and values, and is characterized by usability, scalability and ease of learning.

We demonstrated some of our experiences on using the aforementioned framework and discussed the evaluation of some of the related components. Finally, we demonstrated some of our experiences on using the aforementioned framework for mapping and transforming data from heterogeneous sources to CIDOC CRM and its extensions in various projects. To the best of our knowledge, the X3ML framework is the most complete mapping framework existing today with many tools supporting all different tasks involved.

6.2 Future Work

There are many directions that we are currently exploring or plan to work on in the immediate future. First of all, as already discussed, most of the components are under continuous development and testing. More specifically we are working on making the X3ML framework more generic and scalable.

Moreover, an important future effort will be the consideration of alternative types of sources (at least for the X3ML Engine component). As already described in §4.7, the current version of the X3ML Engine requires the source records to be expressed as an XML document, but our plan is to extend it to support other types of input, such as, in particular, RDF input. This requires several modifications in the design and implementation of the engine. More importantly, the basic construct that we use for reading the source data will be an RDF model (e.g., Jena, Sesame), so instead of XPATH queries the SPARQL [54] language will be used for querying. Furthermore, the Instance Generator component should be able to carry the URIs from the source records to the target records if needed.

One apparent advantage of this approach is that the framework will support input and output of the same format. This sparked the light to investigate another direction: that of *invertible X3ML mappings*. In an invertible X3ML mapping, one can identify in a unique manner (and consequently regenerate) the data in the source dataset that led to the creation of each piece of data in the target dataset. Based on this idea, we can formalize the notion of *invertibility* by trying to identify how X3ML maps the source data to the target data.

Invertibility requires regenerating both the data, and the URIs. For the latter, the association table (that was mentioned earlier in §4.7) can be used to support this functionality. The former is more difficult and consists in determining the triple(s) in the source dataset that contributed in the generation a given (set of) triple(s) in the target dataset. Note that it is not always possible to determine that; in some cases, a triple in the target dataset can be (potentially) generated by two (or more) different sets of triples in the source dataset, therefore the source cannot be uniquely determined. Defining necessary and sufficient conditions for invertibility, and identifying the inverse X3ML mapping (when one exists), is ongoing work; a preliminary discussion appears in [52].

Solving the basic problem of invertibility will also allow addressing more complex problems that are interesting from the curator's point of view, such as:

1. Given a set of target records that have been derived from a set of source records using the appropriate X3ML mappings, and some updates that are applied over the target records, how can one propagate the necessary changes back to the source records?
2. Consider an integrated target dataset, that has been aggregated from the application of multiple sets of X3ML mappings upon multiple source records. Suppose that the aggregation of information from the different records allows the inference of additional information, not present in (or inferrable by) any of the source records. How can this inferred information be propagated back to the source records, and how can one determine the sources that will get each new information?

Finally as the number of mappings grow, it is becoming important to categorize them for assisting users finding easily the desired mappings within the 3M component. Apart from the search and filter functionalities that are already available, we plan to support also a categorization of mappings with respect to different thematic categories.

Acknowledgements

This work was partially supported by the following projects: *ARIADNE* (FP7 Research Infrastructures, 2013-2017), *PARTHENOS* (H2020 Research Infrastructures, 2015-2019), *BlueBRIDGE* (H2020 Research Infrastructures, 2015-2018), and *VRE4EIC* (H2020 Research Infrastructures, 2015-2018). The authors would also like to thank Nikos Anyfantis for working with the Source and Target Analyzer components, and Korina Doerr for designing the user interfaces of the X3ML framework.

References

1. ARIADNE - Advanced Research Infrastructure for Archaeological Dataset Networking in Europe, FP7 Research Infrastructures, 2013-2017. http://www.ariadne-infrastructure.eu/.
2. AstroGrid-D: A Transformation from XML to RDF via XSLT. http://www.gac-grid.de/project-products/Software/ XML2RDF.html (accessed on December 2015).
3. DataTank: Transform datasets into a RESTful API. http://thedatatank.com/ (accessed on December 2015).
4. Groovy: A multi-faceted language for the Java platform. http://www.groovy-lang.org/ (accessed on December 2015).
5. ITN-DCH, Initial Training Network for Digital Cultural Heritage, 2013-2017. http://www.itn-dch.eu/.
6. LifeWatch Greece - National Strategic Reference Framework, 2012-2015. https://www.lifewatchgreece.eu/.
7. Openlink Software: Mapping Relational Data to RDF with Virtuoso's RDF Views. http://virtuoso.openlinksw.com/whitepapers/relational %20rdf%20views%20mapping.html (accessed on December 2015).
8. OpenRefine. http://openrefine.org/ (accessed on December 2015).
9. RDB2RDF Implementations. http://www.w3.org/2001/sw/rdb2rdf/wiki/Implementations (accessed on December 2015).
10. RDFizers. http://wiki.opensemanticframework.org/index.php/ RDFizers (accessed on December 2015).
11. Sip Creator. https://github.com/delving/delving/tree/ master/sip-creator (accessed on December 2015).
12. SquirrelRDF. http://jena.sourceforge.net/SquirrelRDF/ (accessed on December 2015).
13. Tarql: SPARQL for Tables. https://github.com/tarql/tarql (accessed on December 2015).
14. Tripliser. http://daverog.github.io/tripliser/ (accessed on December 2015).
15. Vertere RDF. https://github.com/knudmoeller/Vertere-RDF (accessed on December 2015).
16. Virtuoso Sponger. http://virtuoso.openlinksw.com/ dataspace/doc/dav/wiki/Main/VirtSponger (accessed on December 2015).
17. VRE4EIC - A Europe-wide Interoperable Virtual Research Environment to Empower Multidisciplinary Research Communities and Accelerate Innovation and Collaboration, H2020 Research Infrastructures 2015-2018. http://www.vre4eic.eu/.
18. dFMRÖ - digitale Fundmünzen der Römischen Zeit in Österreich. Available from: http://www.oeaw.ac.at/antike/index.php?id=358, 2007.
19. A. Asserson, K. G. Jeffery, and A. Lopatenko. CERIF: past, present and future: an overview. *euroCRIS*, 2002.
20. S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: light-weight linked data publication from relational databases. In *Proceedings of the 18th international conference on World wide web*, pages 621–630. ACM, 2009.
21. J. Barrasa, O. Corcho, and A. Gómez-Pérez. Fund finder: A case study of database-to-ontology mapping. In *Semantic Integration Workshop*, page 9. Citeseer, 2003.
22. J. Barrasa Rodríguez, Ó. Corcho, and A. Gómez-Pérez. R2O, an extensible and semantically based database-to-ontology mapping language. 2004.
23. T. Berners-Lee. Relational databases on the semantic web. 2013.
24. T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

25. S. Bischof, S. Decker, T. Krennwallner, N. Lopes, and A. Polleres. Mapping between RDF and XML with XSPARQL. *Journal on Data Semantics*, 1(3):147–185, 2012.
26. C. Bizer. D2R MAP - a database to RDF mapping language. *WWW (Posters)*, 2003.
27. C. Bizer and A. Seaborne. D2RQ - treating non-RDF databases as virtual RDF graphs. In *Proceedings of the 3rd international semantic web conference (ISWC2004)*, volume 2004. Citeseer Hiroshima, 2004.
28. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In *KR*, pages 2–13, 1998.
29. E. Coburn, R. Light, G. McKenna, R. Stein, and A. Vitzthum. LIDO-lightweight information describing objects version 1.0. *ICOM International Committee of Museums*, 2010.
30. D. Connolly et al. Gleaning resource descriptions from dialects of languages (GRDDL). *W3C, W3C Recommendation*, 11, 2007.
31. S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF mapping language. 2012.
32. C. P. de Laborda and S. Conrad. Relational.OWL: a data and schema representation format based on OWL. In *Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling-Volume 43*, pages 89–96. Australian Computer Society, Inc., 2005.
33. A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. RML: a generic language for integrated RDF mappings of heterogeneous data. In *Proceedings of the 7th Workshop on Linked Data on the Web (LDOW2014), Seoul, Korea*, 2014.
34. M. Doerr. CIDOC-CRM family of models. http://www.ics.forth.gr/isl/CRMext.
35. M. Doerr. The CIDOC conceptual reference module: an ontological approach to semantic interoperability of metadata. *AI magazine*, 24(3):75, 2003.
36. M. Doerr, S. Gradmann, S. Hennicke, A. Isaac, C. Meghini, and H. van de Sompel. The europeana data model (EDM). In *World Library and Information Congress: 76th IFLA general conference and assembly*, pages 10–15, 2010.
37. W. W. Eckerson. Three tier client/server architectures: achieving scalability, performance, and efficiency in client/server applications. *Open Information Systems*, 3(20):46–50, 1995.
38. A. Felicetti, P. Gerth, C. Meghini, and M. Theodoridou. Integrating heterogeneous coin datasets in the context of archaeological research. *Workshop for Extending, Mapping and Focusing the CRM - co-located with TPDL'2015*, September 2015.
39. A. Felicetti, T. Scarselli, M. Mancinelli, and F. Niccolucci. Mapping ICCD archaeological data to CIDOC-CRM: the RA schema. *A Mapping of CIDOC CRM Events to German Wordnet for Event Detection in Texts*, page 11, 2013.
40. J. Gregorio, R. Fielding, M. Hadley, M. Nottingham, and D. Orchard. RFC 6570: URI template. *Internet Engineering Task Force (IETF) Request for Comments*, 2012.
41. M. Hert, G. Reif, and H. C. Gall. A comparison of RDB-to-RDF mapping languages. In *Proceedings of the 7th International Conference on Semantic Systems*, pages 25–32. ACM, 2011.
42. H. Kondylakis, M. Doerr, and D. Plexousakis. Mapping language for information integration. *Technical Report ICS-FORTH*, 385, 2006.
43. H. Kondylakis, G. Flouris, and D. Plexousakis. Ontology and schema evolution in data integration: Review and assessment. In *On the Move to Meaningful Internet Systems: OTM 2009, Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009, Vilamoura, Portugal, November 1-6, 2009, Proceedings, Part II*, pages 932–947, 2009.
44. H. Kondylakis and D. Plexousakis. Exelixis: evolving ontology-based data integration system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 1283–1286, 2011.
45. H. Kondylakis and D. Plexousakis. Ontology evolution: Assisting query migration. In *Conceptual Modeling - 31st International Conference ER 2012, Florence, Italy, October 15-18, 2012. Proceedings*, pages 331–344, 2012.
46. H. Kondylakis, D. Plexousakis, V. Hrgovcic, R. Woitsch, M. Premm, and M. Schüle. Agents, models and semantic integration in support of personal ehealth knowledge spaces. In *Web Information Systems Engineering - WISE 2014 - 15th International Conference, Thessaloniki, Greece, October 12-14, 2014, Proceedings, Part I*, pages 496–511, 2014.
47. H. Kondylakis, E. G. Spanakis, S. Sfakianakis, V. Sakkalis, M. Tsiknakis, K. Marias, X. Zhao, H. Yu, and F. Dong. Digital patient: Personalized and translational data management through the myhealthavatar EU project. In *37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC 2015, Milan, Italy, August 25-29, 2015*, pages 1397–1400, 2015.
48. C. Lange. Krextor-an extensible framework for contributing content math to the web of data. In *Intelligent Computer Mathematics*, pages 304–306. Springer, 2011.
49. A. Langegger and W. Wöß. *XLWrap–querying and integrating arbitrary spreadsheets with SPARQL*. Springer, 2009.
50. B. Lavoie. Meeting the challenges of digital preservation: The oais reference model. *OCLC Newsletter*, 243:26–30, 2000.
51. Y. Marketakis, Y. Tzitzikas, C. Tona, M. Argenti, F. Marelli, M. Albani, R. Guarino, B. Polsinelli, and R. Bitto. On harmonizing earth science policies, semantics, metadata and ontologies. *Ensuring the Long-Term Preservation and Value Adding to Scientific and Technical Data (PV'2013)*, November 2013.
52. N. Minadakis, Y. Marketakis, H. Kondylakis, G. Flouris, M. Theodoridou, M. Doerr, and G. de Jong. X3ML framework: An effective suite for supporting data mappings. *Workshop for Extending, Mapping and Focusing the CRM - co-located with TPDL'2015*, September 2015.
53. M. J. O'Connor, C. Halaschek-Wiener, and M. A. Musen. Mapping master: A flexible approach for mapping spreadsheets to OWL. In *The Semantic Web–ISWC 2010*, pages 194–208. Springer, 2010.
54. E. Prud'Hommeaux, A. Seaborne, et al. Sparql query language for rdf. *W3C recommendation*, 15, 2008.
55. F. Scharffe, G. Atemezing, R. Troncy, F. Gandon, S. Villata, B. Bucher, F. Hamdi, L. Bihanic, G. Képéklian, F. Cotton, et al. Enabling linked data publication with the datalift platform. In *Proc. AAAI workshop on semantic cities*, pages No–pagination, 2012.
56. F. Simeoni, L. Candela, G. Kakaletris, M. Sibeko, P. Pagano, G. Papanikos, P. Polydoras, Y. Ioannidis, D. Aarvaag, and F. Crestani. *A grid-based infrastructure for distributed retrieval*. Springer, 2007.

57. O. Smyrnaki. Design and implementation of a semi-automatic tool for mapping source schemas to target ontologies. Master's thesis, University of Crete, Voutes Campus, 70013 Heraklion, 2013.
58. P. Szekely, C. A. Knoblock, F. Yang, X. Zhu, E. E. Fink, R. Allen, and G. Goodlander. Connecting the smithsonian american art museum to the linked data cloud. In *The Semantic Web: Semantics and Big Data*, pages 593–607. Springer, 2013.
59. D. Thomas and A. Hunt. Orthogonality and the DRY principle, 2010.
60. Y. Tzitzikas, C. Allocca, C. Bekiari, Y. Marketakis, P. Fafalios, M. Doerr, N. Minadakis, T. Patkos, and L. Candela. Integrating heterogeneous and distributed information about marine species through a top level ontology. In *Metadata and Semantics Research*, pages 289–301. Springer, 2013.
61. Y. Tzitzikas, C. Allocca, C. Bekiari, Y. Marketakis, P. Fafalios, M. Doerr, N. Minadakis, T. Patkos, and L. Candella. Unifying heterogeneous and distributed information about marine species through the top level ontology MarineTLO. *Program: electronic library and information systems*, 50(1), 2015.