# User Manual

## Mapping Analyzer (Maze)

**Nikos Anyfantis (nanifant@ics.forth.gr)**
**01-Feb-16**

# Table of Contents

# List of Figures

# 1  INTRODUCTION

## 1.1  WHAT IS MAZE?

We introduce **Ma**pping Analy**zer** (**Maze)**, a web-based tool which undertakes to serve experts in order to provide a complete management of mappings. Maze works as intermediary between expert users and X3ML language, providing a complete analysis for converting and publishing content as linked data. This tool was implemented in the context of MSc thesis at the University of Crete, School of Sciences and Engineering, Computer Science Department and has been supported by the Foundation for Research and Technology – Hellas (FORTH), Institute of Computer Science (ICS).

Considering the mapping process, we define the main purposes of Maze's usage regarding the management of mappings. In brief, these are:

- **Create a Mapping**: It regards the process in which experts wants to create a mapping from scratch.
- **Improve a Mapping**: It contributes to mapping optimization such as check its quality, compare with others, monitor its evolution etc.

Maze provides 11 main functionalities regarding analysis of source and target schema, percentage coverage metrics, mapping representations, comparison of mappings and tracking changes over time. More specifically, the analysis of schemas offers the opportunity to explore and understand better their content providing multiple capabilities. The percentage coverages of schemas include two different metric methods that allow us to have an overview of the used data. Also, user may exclude data which are considered undesirable for a mapping and calculate new coverage metrics. The graphical representation techniques are divided into two functionalities. The first represents a graph based on language X3ML focusing on syntactic elements of language. The second functionality transforms the schemas in a three-dimensional space creating relationships and illustrating the correlations regarding the selected mapping. As regards the comparison of mappings, we implemented comparisons about the syntax of X3ML language, the graphical representation of correlations and comparison of the generated instances. Finally, the functionality of tracking the evolution of mapping helps the user understand the basic points which have changed over time.

Furthermore, Mapping Analyzer constitutes an integrated system which has been developed with respect to the following software design principles.

a) **Organizational structure**. An important feature of software is the orchestration of background components. All parts are in a structured form, creating an effective collaboration between front-end and back-end environment.

b) **Single responsibility principle**. Each component handles only one responsibility regarding the functionalities. On future if we need to make one change we are going to make it in the component which handles it.

c) **Open/closed principle**. This principle states "*software entities should be open for extension, but closed for modification*"[1] that is, such an entity can allow its behavior to be extended without modifying its source code.

d) **Dependency Inversion Principle**. It refers to a specific form of decoupling software modules. The conventional dependency relationships established from high-level, policy-setting modules to low-level, dependency modules are reversed.

e) **Simplicity**. Last but not least, simplicity and clarity should be achieved by tools in order to   make clear meanings and functions.

Finally, Maze has been originally implemented and integrated in the context of Mapping Memory Manager (3M) and 3M Editor. The actors are able to use this tool through either the "Mapping Analysis" or "Compare Mappings" tab. Special feature is the direct access to all functionalities in real time during the mapping process. This implementation is maintained by the community on GIT repository[2], while a public implementation is available (http://139.91.183.3/Maze/ ).

## 1.2 WHO SHOULD USE MAZE?

An actor is a person or other entity being specified externally to the software system who interacts with the system and performs use cases to accomplish tasks. Different actors often correspond to different user classes, or roles, identified from the users' community. We define three main actors who interact with Maze: a) Source Schema Expert, b) Target Schema Expert, and c) IT Expert. An extended description of these actors follows.

- *Source Schema Expert*. The *Source Schema Expert* is the actor who is responsible for the source schema and he has little or no programming-level knowledge. He knows the describing concept, the dataset and the structure in general of the source schema. Also, he is able to understand the semantics of the concept and the existing links by knowing correlations of existing elements. He knows the variables of the elements and their content (e.g. a numeric field for weight can be in kilograms or pounds). Furthermore, the Source Schema Experts are able to explain easily a given cell but it is very difficult to create a mapping. Usually, these particular actors are the creators or managers of the source datasets.

- *Target Schema Expert*. The *Target Schema Expert* is the actor responsible for the target schema (ontology) and has little or no programming-level knowledge. The majority of target schema experts use a more widely known standard schema, but there is also a growing trend in the linked data world towards custom ontologies. Also, these experts are aware of the principles of the Semantic Web and technologies such as RDF/S and OWL. He is an expert regarding the identification of classes, properties and their specifications. He knows the semantics of ontologies and he is able to map correctly a given concept to the target schema.

---

[1] Meyer, Bertrand (1988). Object-Oriented Software Construction. Prentice Hall.
[2] GitHub repository: https://github.com/isl/Maze

- **IT Expert**. The *IT Expert* has high programming-level knowledge but he has no experience regarding the target and source schemata. Usually, he is aware of used technologies about the schemata but he is not able to create a mapping since he lacks of experience. However, he is more effective in URI generation process since he knows the mapping language. In our case, we assume that the main mapping language is X3ML and for this reason the IT Expert can be named X3ML Expert.

# 2  3M CONTEXT

## 2.1 X3ML LANGUAGE

The X3ML is an XML based language that describes mappings which can be created and discussed collaboratively by experts. Until now, the mappings were created manually. For this reason, the X3ML emphasizes on a standardized description of a mapping which aims to a better cooperation and creation of mapping memory in order to accumulate knowledge and experience.

The X3ML language designed and implemented by the Foundation for Research and Technology (FORTH) in 2006. Primarily, it was designed to follow the principle DRY (avoiding repetition) and be clearer about the URI generating process.

X3ML separates schema mapping from the concern of generating proper URIs so that different expertise can be applied these two very different responsibilities. The URI expert must ensure that the generated URIs match certain criteria such as consistency and uniqueness, while the Schema experts only need to concern themselves with the proper interpretation of the source.

### Global Structure

By studying the global structure of the X3ML language, we could say that it is quite understandable and properly structured. The general format is consisted of a specific shape and it provides a series of relations between the source and target schemata.

Below we can see the general form of X3ML. Each X3ML mapping contains a root element with name "*x3ml*". It contains a "*namespaces*" and a "*mappings*" element. The first includes all namespaces that we can use in the mapping. The second element includes one or more "children" elements which express the transformation rules. The X3ML language uses the XML language in order to keep a structured form and XPath to express selectors. These are the main technologies in order to create a mapping from scratch.

```
<x3ml>
    <namespaces/>
    <mappings>
        <mapping>
            <domain>
                <source_node/>
                <target_node/>
            </domain>
            <link>
                <path>
                    <source_relation/>
                    <target_relation/>
                </path>
                <range>
                    <source_node/>
                    <target_node/>
                </range>
            </link>
            ... more links ...
        </mapping>
        ... more mappings ...
    </mappings>
</x3ml>
```

Each mapping rule is expressed in a "*mapping*" element where each of them contains the elements *domain* and *range*. These elements include *target nodes* which can either include or create URIs or represent literals to specify the related details of the ontology. The target blocks can also include criteria which restrict a mapping in order to meet the requirements of the transformation and may permit the direct creation of nodes and relationships.

Each class-property-class of the source schema is mapped individually to the global schema and can be seen as self-explanatory, context independent proposition. 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

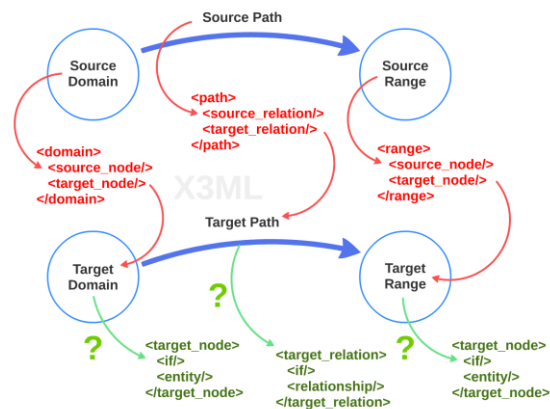Below we can see the general form of a mapping and its features.



**Figure 1 Structure of X3ML**

## 2.2 MAPPING MEMORY MANAGER (3M)

The 3M web application is an open source schema mapping tool developed by a consortium through an ongoing project called CultureBroker. A public implementation of 3M is available from the FORTH web site http://www.ics.forth.gr/isl/3M/ . FORTH (Foundation for Research & Technology – Hellas) is the main developer of 3M and is also the administrative home of the CIDOC CRM (Conceptual Reference Model). Delving BV (www.delving.eu) is a software development company located in Holland and has contributed to the development of the X3ML engine that handles the URI generation and the data transformation to RDF.

3M allows data experts to transform their internal structured data and other associated contextual knowledge to other schemas and, in particular, the CIDOC CRM (Conceptual Reference Model). Fields or elements from a source database (Source Nodes) are aligned with one or more entities described in the target schema so that the data

from an entire system can be transformed. The purpose of this is typically for publication on the Web and in particular meaningful integration with other data also transformed to the same target schema.

Although 3M can map data to other schemas it is particularly aimed at mapping to contextual data and to CIDOC CRM, and alignment to it is based, not just on Source Nodes, but on other implicit information known to the owning organization. Mapping data requires good communication with the producers of data because mapping, particularly to the CIDOC CRM, is not simply a technical exercise but provides the opportunity for additional knowledge to be encoded and published. The development of 3M attempts to support the need to involve and record the knowledge of the data producers and ensure the best possible semantic and contextual mapping of cultural data.

3M is part of an ongoing development project based on a Data Provisioning reference model called Synergy (http://www.cidoc-crm.org/docs/SRM_v1.4.pdf ). Synergy describes the relationships and processes required for high quality data provisioning and data integration. It identifies the different processes and roles involved in data provisioning and data aggregation. 3M supports some of these processes but in particular the schema matching and X3ML transformation, including support for Linked Data Publication.



**Figure 2 Mapping Memory Manager**

## 2.3 3M EDITOR

The 3M Editor combines the X3ML language and the X3ML Engine[3], providing a Graphical User Interface (GUI) in order to facilitate the total mapping process. It provides an easier way to create mappings graphically and it is connected with the engine to generate instantly triples from the source to the target schema.

---

[3] X3ML engine handles the URI generation and data transformation to rdf.

As we described in previous subsection, the initial form of X3ML is very difficult to be processed by non-experts. Therefore, FORTH has developed a platform in order to manipulate and create mappings graphically. The new form of mappings is displayed as a table.

Also, the X3ML Engine requires an IT expert to import the source, the target schema and the mappings in order to generate the RDF format. The 3M Editor can facilitate this process by connecting the mappings with the engine automatically.

Overall, the main functionalities of the 3M Editor provide the user with the capability to:

- Provide info about the mappings
- Import Source Schema in XML format
- Import Target Schema (e.g. RDF/S format)
- Be informed about suggestions during the mapping process based on the schemata
- Create mapping graphically
- See and edit the initial form of X3ML
- Define and Generate URIs
- Use the X3ML Engine in order to generate the final result



Figure 3 3M Editor

# 3 Using Maze

## 3.1 Installation

Since Maze has been developed as Maven project, it provides all required libraries in **pom.xml**. Also, the folder **src** contains all the files needed to build the web app and create a **war** file. Therefore, the installation requires only four steps:

1. **Download**. You can download this project from GitHub repository (https://github.com/isl/Maze).
2. **Build**. To compile the source files and package the application, open this project with an IDE (i.e. Netbeans) and build with dependencies.
3. **Deploy**. You may use any application server that supports war files (recommended: Apache Tomcat).
4. **Run**. In order to launch application, you only need to open the server's URL followed by tool's folder (e.g. http://localhost:8080/Maze/ ).

Moreover, if you need further configuration, you can edit the Controller which is located at **Maze/src/main/webapp/app/js/Controller.js**. For design or further development purposes, you can set Maze in **DebugMode** (true). Also, you may configure ServerURL (hosted url), port, Maze's Rest services URL (url which hosts Maze's server) and target schema service of 3M. Figure 4 illustrates the default configuration.

```
var GlobalResources = {
    System :{
        DebugMode : false,    //no connection with server at all (only for design purposes)
        CacheMode : false     //NOT SUPPORTED YET get data if they are not cached already
    },
    Services : {
        ServerURL: 'http://'+window.location.hostname+':8080',
        RestService: 'http://'+window.location.hostname+':8080/Maze/webresources',
        TargetSchemaService:'http://139.91.183.3/3MEditor/FetchBinFile?type=target_info&file='
    }
};
```

**Figure 4 Configuration File**

Finally, you may set services to retrieve resources (i.e. X3ML files, source and target schema, generated data records, etc.) for server. The following figure illustrates the default configuration. The file is located at **Maze/src/main/webapp/WEB-INF/config.properties.**

```
#URL to retrieve X3ML from 3M service
Service_X3ML=http://139.91.183.3/3MEditor/Services?method=export&output=text/xml&id=

#URL to retrieve XML file from 3M service
Service_SourceSchema=http://139.91.183.3/3MEditor/FetchBinFile?type=xml_link&file=

#URL to retrieve Target schema file from 3M service
Service_TargetSchema=http://139.91.183.3/3MEditor/FetchBinFile?type=target_info&file=

#URL to retrieve Data records (instances) file from service
Service_DataRecords=http://139.91.183.3/3MEditor/FetchBinFile?type=example_data_target_record&file=

#URL to retrieve Versions of mapping
Service_GetVersions=http://139.91.183.3/3M//GetVersionsList?type=Mapping&id=Mapping

#URL to retrieve X3ML mapping from version collection
Service_VersionedX3ML_Part1=http://139.91.183.3/3MEditor/Services?method=export&output=text/xml&id=
Service_VersionedX3ML_Part2=&version=
```

**Figure 5 Services Properties File**

## 3.2 GRAPHICAL USER INTERFACE

The user interface has been implemented based on modern design patterns, following the basic user interface design principles. Also, it has been designed in order to be adapted to mobile devices and to be supported by the majority of modern web browsers such as Google Chrome, Mozilla Firefox, Opera, etc.

The main page is **index.html** which conducts users in order to select one of two main functionalities. Single Mapping Analysis tab contains a textbox in which you can give the unique mapping ID. By clicking **Analyze** you are transferred to single mapping analysis. Comparison of Mappings allows you to give two different mapping IDs and click **Compare** in order to compare them.
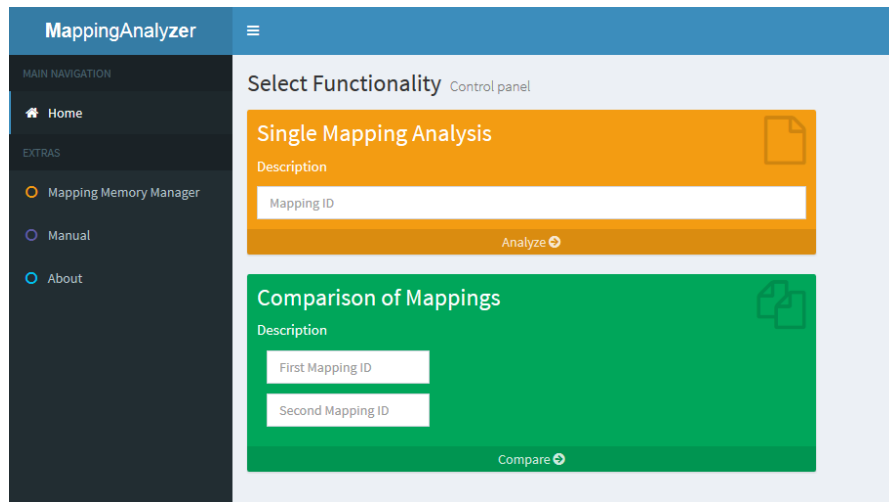


**Figure 6 Index page**

Other important pages are **about.html** and **errorpage.html** (Figure 7). The first contains information about Maze such as license conditions, description, contact info, GitHub repository and other resources, while the second is appeared in cases when the mapping ID is incorrect or the requested page is not available.
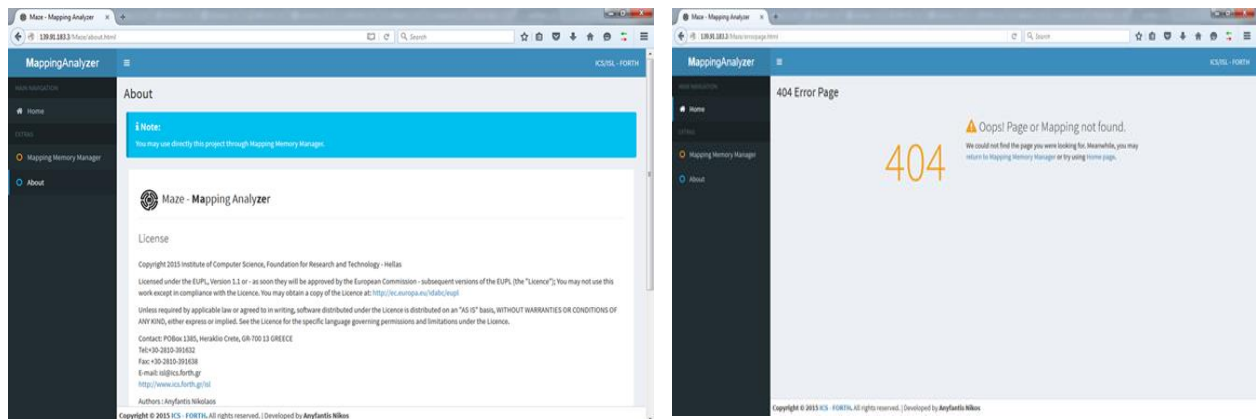


**Figure 7 About and Error pages**

The Mapping Memory Manager (3M) and 3M Editor work as intermediary between users and Maze. You can access this project through Mapping Memory Manager community. More specifically, you may use Maze through:

1. 3M – **More** option contains direct links to single mapping analysis (**Analysis**) or comparison of two mappings (**Compare**). Figure 8 draws both options.
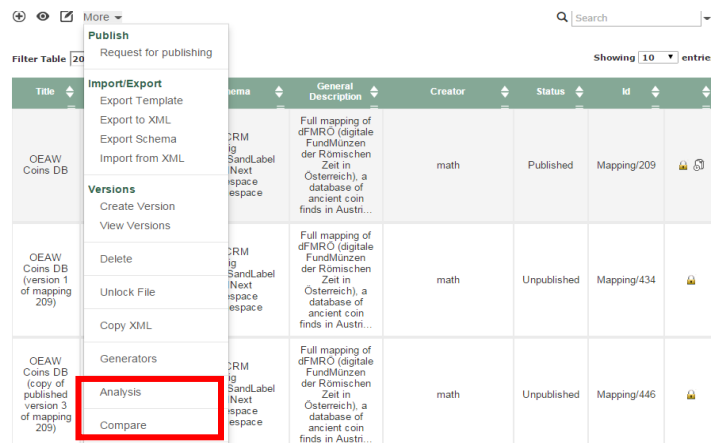


Figure 8 Accessing Maze through 3M

2. 3M Editor – You may view analysis of your mapping by using the **Analysis** tab and clicking the "view Analysis" link.
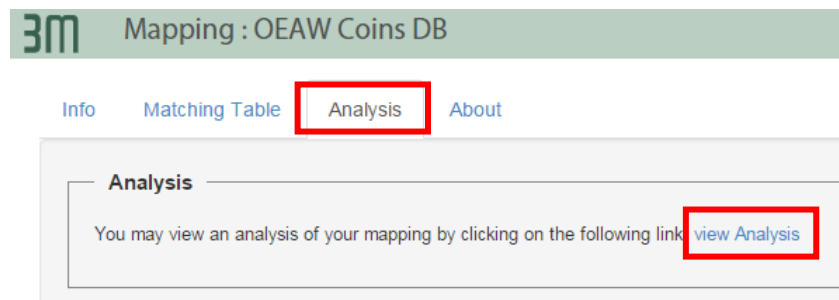


Figure 9 Accessing Maze through 3M Editor

3. Direct link – Finally, you may give directly the mapping ID as parameter in URL as **"?id=Mapping"** and your mapping. For example, in Figure 10 we give the ID **209** in order to be analyzed.
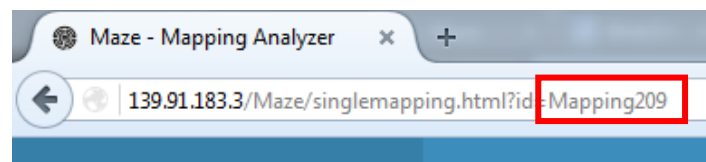


Figure 10 Direct URL

## 3.3 MAZE WEB SERVICES

Despite the graphical user interface, the Maze implementation has been developed in order to provide a complete API of all provided functionalities. It was designed as a REST based-Web Service for X3ML mappings using the Java API for Restful Service (JAX-RS API) with Jersey that exposes multiple resources. More precisely, you may consume data in XML or JSON format making simple requests.

Let assume that the implementation has been deployed locally (http://localhost:8080). The complete API is described below:

**URI**        http://localhost:8080/Maze/webresources/x3ml/singlemapping/{id}

| | |
|---|---|
| **Method** | GET |
| **Produces** | XML/JSON |
| **Functionality** | Returns basic X3ML structure depending on requested format |

**URI**        http://localhost:8080/Maze/webresources/x3ml/singlemapping/plain/{id}

| | |
|---|---|
| **Method** | GET |
| **Produces** | XML |
| **Functionality** | Returns X3ML as is, ordering *mapping* and *link* elements. It is used for textual comparison. |

**URI**        http://localhost:8080/Maze/webresources/x3ml/singlemapping/coveragemetrics/{id}

| | |
|---|---|
| **Method** | GET |
| **Produces** | XML/JSON |
| **Functionality** | Returns all coverage metrics of source and target schema of mapping. |

**URI**        http://localhost:8080/Maze/webresources/x3ml/source_schema/er/{id}

| | |
|---|---|
| **Method** | GET |
| **Produces** | XML/JSON |
| **Functionality** | Return in structured format tables and their attributes. |

| URI | http://localhost:8080/Maze/webresources/x3ml/source_schema/tree/{id} |
|---|---|
| **Method** | GET |
| **Produces** | XML/JSON |
| **Functionality** | Returns source schema as tree graph including parent-children relations. |

| URI | http://localhost:8080/Maze/webresources/x3ml/target_schema/all/{id} |
|---|---|
| **Method** | GET |
| **Produces** | XML/JSON |
| **Functionality** | Returns statistics, classes, subclasses, properties, subproperties and other related information of target schemas. |

| URI | http://localhost:8080/Maze/webresources/x3ml/mappingrules/er/{id} |
|---|---|
| **Method** | GET |
| **Produces** | XML/JSON |
| **Functionality** | Returns as structured format mapping rules between source (ER) and target schema (Mapping-centered representation). |

| URI | http://localhost:8080/Maze/webresources/x3ml/mappingrules/tree/{id} |
|---|---|
| **Method** | GET |
| **Produces** | XML/JSON |
| **Functionality** | Returns as structured format mapping rules between source (TREE) and target schema (Mapping-centered representation). |

| URI | http://localhost:8080/Maze/webresources/x3ml/source_schema/coveredelements/{id} |
|---|---|
| **Method** | GET |
| **Produces** | XML/JSON |
| **Functionality** | Returns all covered elements considering the mapping and source schema. |

| URI | http://localhost:8080/Maze/webresources/x3ml/target_schema/coveredelements/{id} |
|---|---|
| **Method** | GET |
| **Produces** | XML/JSON |
| **Functionality** | Returns all covered elements considering the mapping and target schema. |

| URI | http://localhost:8080/Maze/webresources/x3ml/source_schema/metrics/excludinglist/{id} |
|---|---|
| **Method** | POST |
| **Consumes** | TEXT_PLAIN |
| **Produces** | XML/JSON |
| **Functionality** | Consumes plain text as JSON format the requested excluding elements of source schema and returns new percentage coverage metrics. |

| URI | http://localhost:8080/Maze/webresources/x3ml/target_schema/metrics/excludinglist/{id} |
|---|---|
| **Method** | POST |
| **Consumes** | TEXT_PLAIN |
| **Produces** | XML/JSON |
| **Functionality** | Consumes plain text as JSON format the requested excluding classes or properties of target schema and returns new percentage coverage metrics. |

| URI | http://localhost:8080/Maze/webresources/x3ml/instances/{id} |
|---|---|
| **Method** | GET |
| **Produces** | XML/JSON |
| **Functionality** | Returns all instances in structured format, including URI, label, incoming and outgoing properties and type of class. |

| URI | http://localhost:8080/Maze/webresources/x3ml/graphical_comparison/{mappid1}/{mappid2} |
|---|---|
| **Method** | GET |
| **Produces** | XML/JSON |
| **Functionality** | Returns a list of common and different mapping assertions between two mappings. It contributes to graphical comparison. |

| URI | http://localhost:8080/Maze/webresources/x3ml/versions/{id} |
|---|---|
| **Method** | GET |
| **Produces** | XML/JSON |
| **Functionality** | Returns a list of available versions and related information such as id, date, user and comments. |

| URI | http://localhost:8080/Maze/webresources/logs/get |
|---|---|
| **Method** | GET |
| **Produces** | Maze-logfile.log File |
| **Functionality** | Downloads log file of Maze implementation. |

| URI | http://localhost:8080/Maze/webresources/logs/show |
|---|---|
| **Method** | GET |
| **Produces** | TEXT_PLAIN |
| **Functionality** | Displays log file of Maze implementation. |

# 4 SINGLE MAPPING ANALYSIS

Since you have installed the implementation, the first major management panel is the analysis of single mapping. As already described, you only need to give as input a mapping ID and Maze's backend retrieve and analyze the X3ML mapping file automatically. Figure 11 depicts the **Home** page of interface.

In order to be more precise, you may explore all provided pages through the left side bar, providing quick navigation among them. Also, each menu item contains submenus in order to categorize functionalities, while the right side of page is the main content of the page. Home page contains an overview of all functionalities and provides a small description about their capabilities.

- The **Single Mapping** menu item regards source and target schema analysis, warnings and errors.
- The **Mapping Graphs** contains representation of mappings (X3ML-centered and Mapping-centered).
- The **Percentage Coverage** item includes metrics and other information about coverages between involved schemas and mapping.
- **Instance Representation** illustrates graphs about the generated instances after the mapping process. This functionality requires the involvement of X3ML engine.
- Finally, the **Evolution** menu item allows users to track changes about several versions of selected mapping.
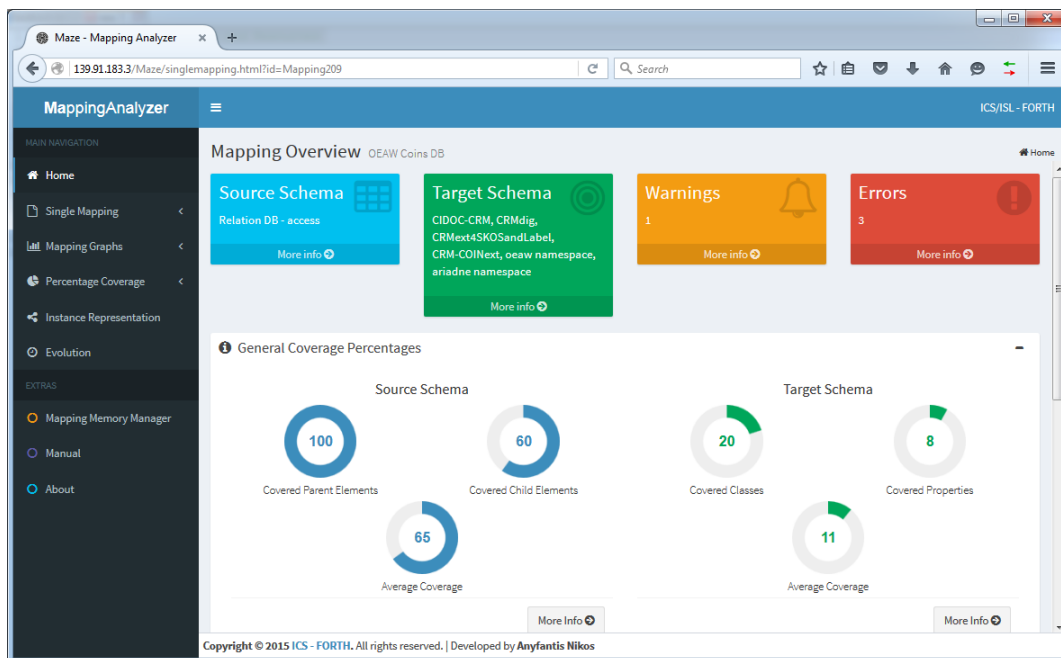


**Figure 11 Home page of single mapping analysis**

## 4.1 EXPLORE SOURCE SCHEMA

This functionality is used to explore the given source schema of mapping. It requires to an uploaded source schema through 3M Editor, which might be an example XML file or an XML schema file. Valid formats are, XML (a data file), XSD (a XML schema file). In cases which the source schema is ER, you only need to export the relational database as XML format.

Firstly, the implementation allows user to be informed about source schema details. It describes the **name**, **type**, **version** and info about **collection**.

| Schema | Type | Version |
| --- | --- | --- |
| Relation DB - access | ER | Not Available |

| Collection |
| --- |
| dFMRÖ (digitale FundMünzen der Römischen Zeit in Österreich) (http://www.oeaw.ac.at/numismatik/projekte/dfmroe/dfmroe.html) |

**Figure 12 Source Schema details**

Furthermore, some additional information can be provided including sample data, for example source schema type, total parent and child elements.

**Sample data**

| Type | Total Parent Elements | Total Child Elements |
| --- | --- | --- |
| ER | 10 | 73 |

**Figure 13 Sample data**

An important capability is the declaration of undesired data for mapping. In most cases source schemas contains private data or data for internal use only. Thus, you can define them searching in parent or child elements. By clicking **Save & Analyze**, you may be informed if these elements have already mapped, while new source schema coverage metrics are available.

**Define undesired data for Mapping**

| Search Fields | country | + |
| --- | --- | --- |
| | • COUNTRY | |
| **Field** | **Covered** | • COUNTRY |
| | • COUNTRY_NAME | |
| | • COUNTRY_ID | |
| | • COUNTRY_NAME_EN | |
| | | Save & Analyze |

**Figure 14 Define undesired data source schema**

The most important functionality is the representation of source schema. Although the XML data is an understandable format for experts, there are complex structures which are difficult to be explored. The representation tries to create a simple graph in order to facilitate users. By default, the generated graph is a tree (**XML**) which allows the capability of search and focus on specific elements by clicking on them. Another representation regards **ER** schemas depicting tables and attributes. Although relations between tables are not

specified in source schema, the graph tracks them automatically and connects them with edges. In cases where there are false derivations the user is able to change them (Figure 15).
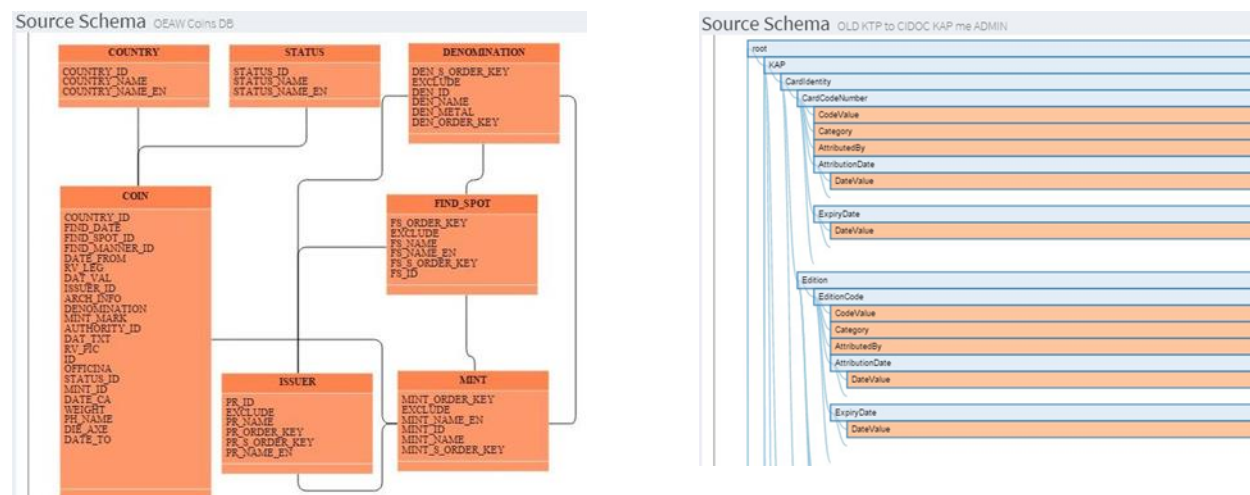


Figure 15 ER / Tree representation of source schema

## 4.2 EXPLORE TARGET SCHEMA

The Explore Target Schema page is used to explore and understand the involved target schemas of mapping. It contains general info, statistics, interactive graphs and other additional options such as exploration of classes and properties.

The **Target Info** tab provides an overview of target schemas. The columns are: a) **Schema** describes the name of source schema, b) **Type** specifies the schema's format, c) **Version** declares the current version, and d) **Schema File** contains direct link to initial schema's format.

ⓘ Target Info

| # | Schema | Type | Version | Schema File |
|---|---|---|---|---|
| 1 | **CIDOC-CRM** | **rdfs** | **6.0** | **cidoc_crm_v6.0-draft-2015January.rdfs** |
| 2 | CRMdig | rdfs | 3.2 | CRMdig_v3.2.rdfs |
| 3 | CRMext4SKOSandLabel | rdfs | 1.2 | CRMext4SKOSandLabel_v1.2.rdfs |
| 4 | CRM-COINext | rdfs | 1.1 | CRM-COINext_v1.1.rdfs |

Figure 16 Selected Target Schema

The Target Schema Statistics container provides information about the selected target schema when you click it from the previous list. **Schema Type** indicates the data format, while there are two number counting total **classes** and **properties** respectively. For example, CIDOC CRM is rdfs type and contains 85 classes and 286 properties.

| **Schema Type** | **Total Classes** | **Total Properties** |
|---|---|---|
| rdfs | 85 | 286 |

Equivalent to source, you are capable to define undesired classes and properties in target schema. You may either search specific classes and properties or load default excluded classes for CIDOC CRM pressing the "Load Default for CIDOC-CRM" button. By clicking **Save & Analyze**, you may be informed if these classes or properties have already used in mapping, while new target schema coverage metrics are available.

Two graphs illustrate relations for classes and properties. Both of them contain hierarchy relations. For example, each class is depicted as a node, while edges indicated its super-classes or sub-classes. Another characteristic is the size of each node. It depends on the total child concepts (sub-classes/sub-properties). Finally, as regards classes' colors are based on official color definitions according to CIDOC CRM.
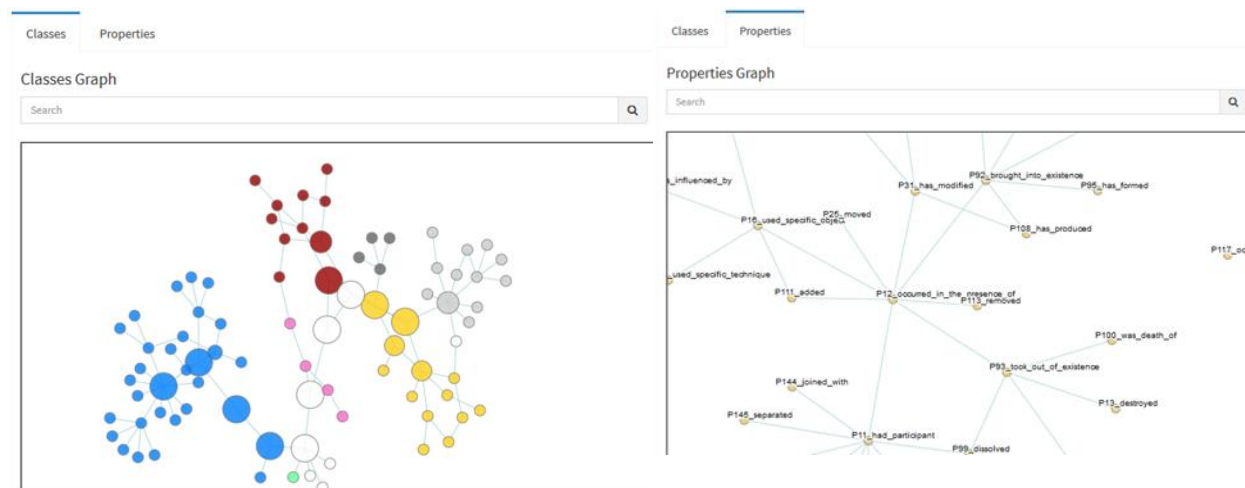
The interactive behavior constitutes a major advantage. You are capable to focus on classes or properties, zoom in/out, select class or even search specific appellations. Figure 20 shows a focused class and its properties on mouse hover.
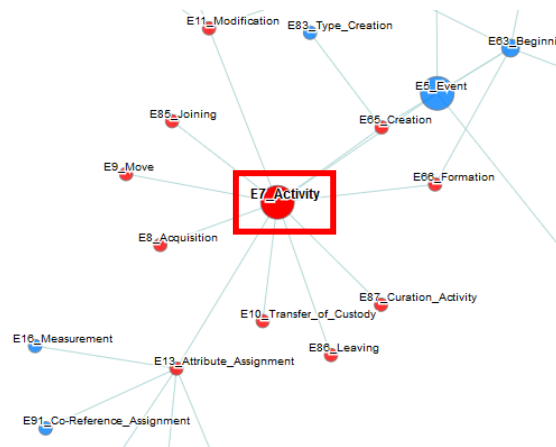


**Figure 20 Focused class and its subproperties**

When you select a class/property, the implementation shows specific information about it. As regards classes, it shows label, URI, Comments and subclasses. Clicking a link of subclass, the graph is updated and focuses on this class. The "**View Properties**" button allows you to explore in detail the selected class.



**Figure 21 Details of focused class**

Below, in Figure 22 you can see incoming and outgoing properties of selected class. There attached labels and arrows which show the properties between classes.
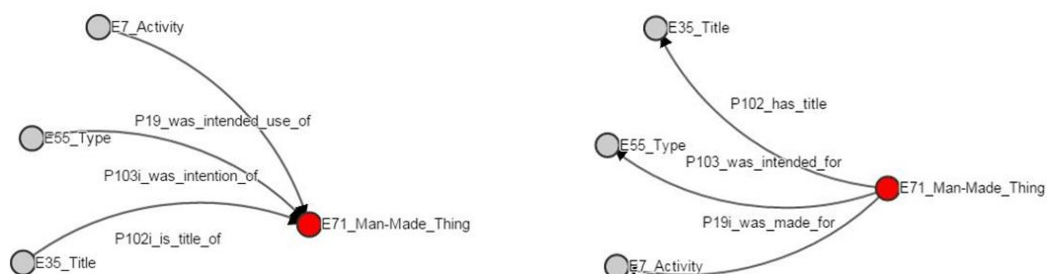


**Figure 22 Incoming and Outgoing Properties of Class**

Explore Target Schema functionality supports triangles of classes. This means that three classes are connected with three properties in a row, where the domain class of first property is the same with range of the third property. The implementation creates a list of triangles (Figure 23) for each class, allowing you to draw each of them.  An indicative example of triangle's graph is depicted in Figure 24.

◁ Triangles                                                                                                                          −

| Property | B Class | Property | C Class | Property | View |
|----------|---------|----------|---------|----------|------|
| http://www.cidoc-crm.org/cidoc-crm/P14_carried_out_by | http://www.cidoc-crm.org/cidoc-crm/E39_Actor | http://www.cidoc-crm.org/cidoc-crm/P11i_participated_in | http://www.cidoc-crm.org/cidoc-crm/E5_Event | http://www.cidoc-crm.org/cidoc-crm/P20i_was_purpose_of | Draw |
| http://www.cidoc-crm.org/cidoc-crm/P17_was_motivated_by | http://www.cidoc-crm.org/cidoc-crm/E1_CRM_Entity | http://www.cidoc-crm.org/cidoc-crm/P2_has_type | http://www.cidoc-crm.org/cidoc-crm/E55_Type | http://www.cidoc-crm.org/cidoc-crm/P21i_was_purpose_of | Draw |
| http://www.cidoc-crm.org/cidoc-crm/P17_was_motivated_by | http://www.cidoc-crm.org/cidoc-crm/E1_CRM_Entity | http://www.cidoc-crm.org/cidoc-crm/P2_has_type | http://www.cidoc-crm.org/cidoc-crm/E55_Type | http://www.cidoc-crm.org/cidoc-crm/P125i_was_type_of_object_used_in | Draw |

**Figure 23 List of triangles**
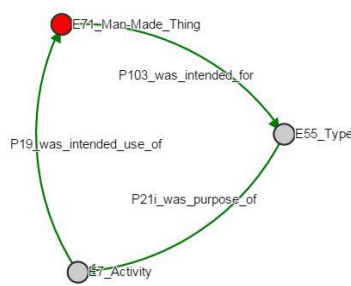


**Figure 24 Graph of triangle**

# 4.3 PERCENTAGE COVERAGE OF SOURCE SCHEMA

Since you have created a mapping, there are several metric about coverage. This functionality permits to identify percentage coverages regarding source schema. Users should be informed about the coverage of the entire source schema or be sure that he has covered the elements with high interest.

The first graph contains three different metrics related to the source schema (Figure 25). 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.
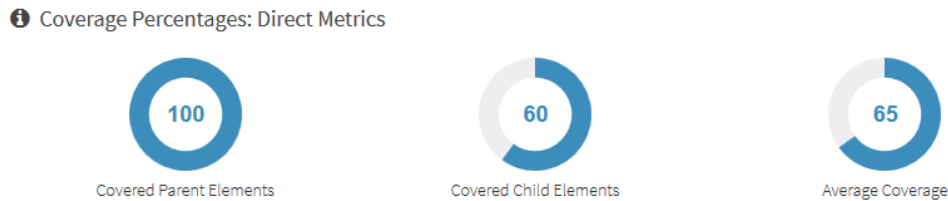


**Figure 25 Metrics of source schema**

Moreover, the system depicts the source schema as tree (XML form) highlighting the covered elements. Figure 26 draws an indicative example. Highlighted and red bordered elements are considered covered, while transparent elements are not mapped.
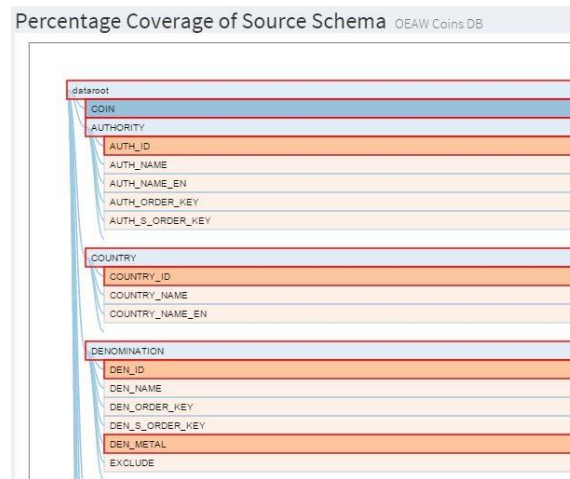


Figure 26 Covered elements source schema

## 4.4 PERCENTAGE COVERAGE OF TARGET SCHEMA

The Target Analyzer allows users to explore the target schema, offering an effective schema analysis. The metrics are divided into two categories: (a) Direct and (b) 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.
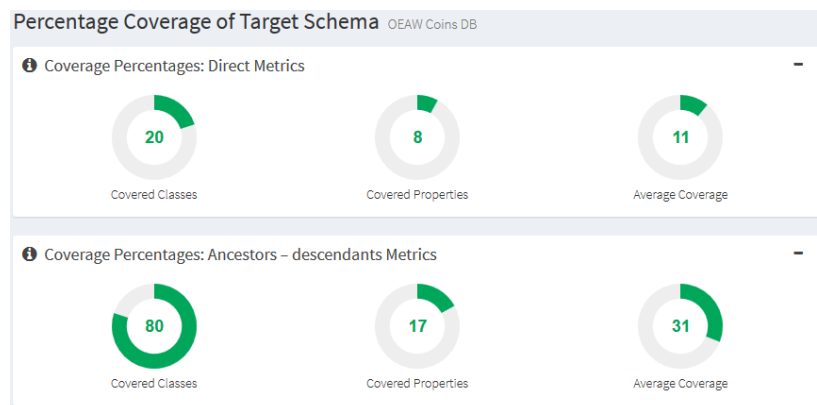


Figure 27 General Metrics of target schema

Figure 27 regards metrics about total target schemas, while Figure 28 depicts specific metrics for each involved target schema. You are capable to click each schema from the list and explore coverages about classes, properties and total average of selected schema.
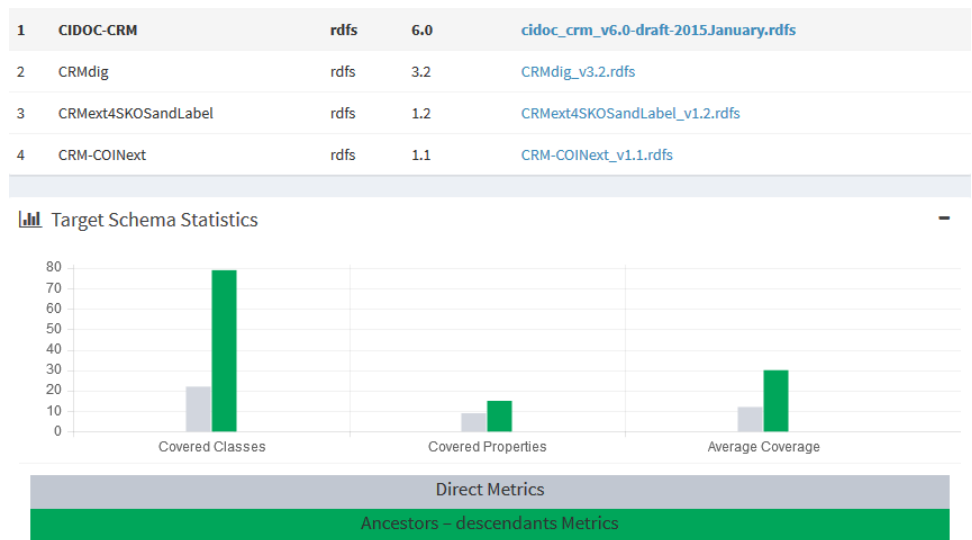
| 1 | **CIDOC-CRM** | **rdfs** | **6.0** | **cidoc_crm_v6.0-draft-2015January.rdfs** |
|---|---|---|---|---|
| 2 | CRMdig | rdfs | 3.2 | CRMdig_v3.2.rdfs |
| 3 | CRMext4SKOSandLabel | rdfs | 1.2 | CRMext4SKOSandLabel_v1.2.rdfs |
| 4 | CRM-COINext | rdfs | 1.1 | CRM-COINext_v1.1.rdfs |



**Figure 28 Metrics of selected target schema**

Accordingly to source schema, there are two graphs about classes and properties, which illustrate covered/uncovered classes and properties respectively. Highlighted and red bordered nodes are considered covered, while transparent elements are not mapped (Figure 29). Finally, all covered classes and properties are depicted as lists.
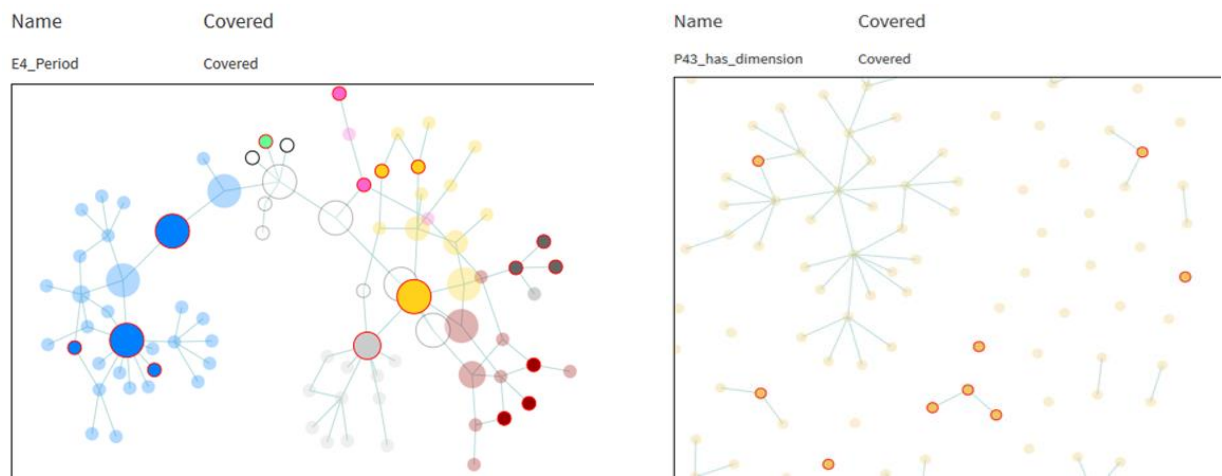


**Figure 29 Covered classes and properties**

## 4.5 X3ML-CENTERED REPRESENTATION

The implementation provides a graph with all mapping rules focusing on syntactic elements of the X3ML language. This particular representation depicts each mapping as a node. As regards the *mapping* nodes, the graph shows the *source_node* and *target_node* which are connected with the main mapping. Finally, we are able to navigate through links which are linked to each mapping. Figure 30 illustrates an indicative example of this particular representation.

Also, you may filter nodes through three buttons "TargetNodes", "SourceNodes" and "Links", while zoom is available to facilitate representation purposes.
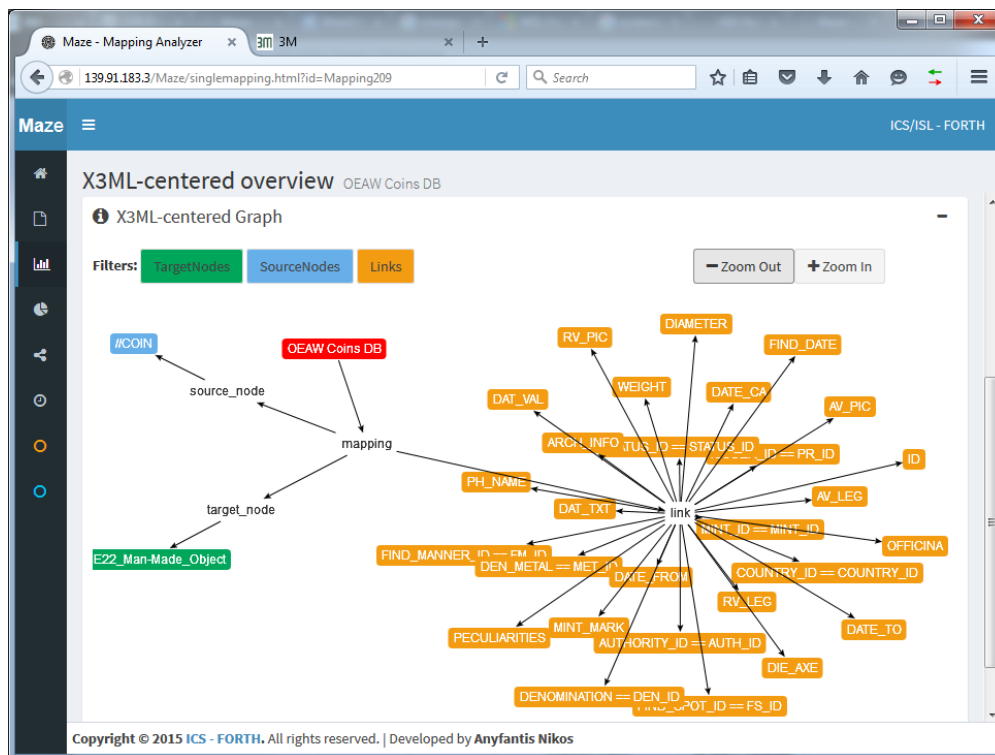


Figure 30 X3ML-centered representation

## 4.6 MAPPING-CENTERED REPRESENTATION

The Mapping-centered representation offers an interactive three-dimensional world that displays source, target and mapping assertions between them. You may explore scene making zoom in/out and select objects.

Firstly, you may explore source schema which is depicted at floor. Two representations are available depending on the type of schema (ER/XML). Figure 31 draws the initial representation of an ER source schema. Tables are depicted as big squares at floor of scene, while small cubes are their attributes. Each shape contains a label that describes their names. For example, the table "COIN" has attributes "COUNTRY_ID", "FIND_DATE" etc. The representation of XML source schema is depicted as an interactive tree structure that is been expanded based on

parent-children relations. Moreover, coverage information is available depending on objects' format. An element is considered as covered if its structure is not-wireframe, while uncovered elements are depicted as wireframe object.



Figure 31 Source schema 3D representation

Each source schema object allows you to view mapping assertion regarding the target schema. By clicking on it, we create each related resource as sphere and connect them with properties. Below we selected the "COIN" table and it is created the "E22_Man-made_Object" sphere. This indicates that there is a mapping rule which translates each row of "COIN" table as this particular class. Also, you can see properties between classes based on target schema. The color is selected regarding the specific color-palette of CIDOC-CRM.



Figure 32 Mapping assertions of source and target schema
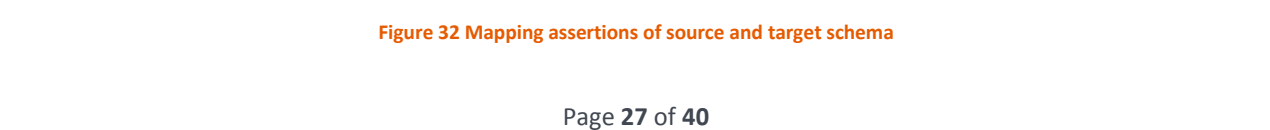
Another view of the same mapping is illustrated in Figure 33. The system draws a subgraph of target schema which includes involved classes regarding the mapping. Also, we are able to identify incoming and outgoing properties, creating edges between them.
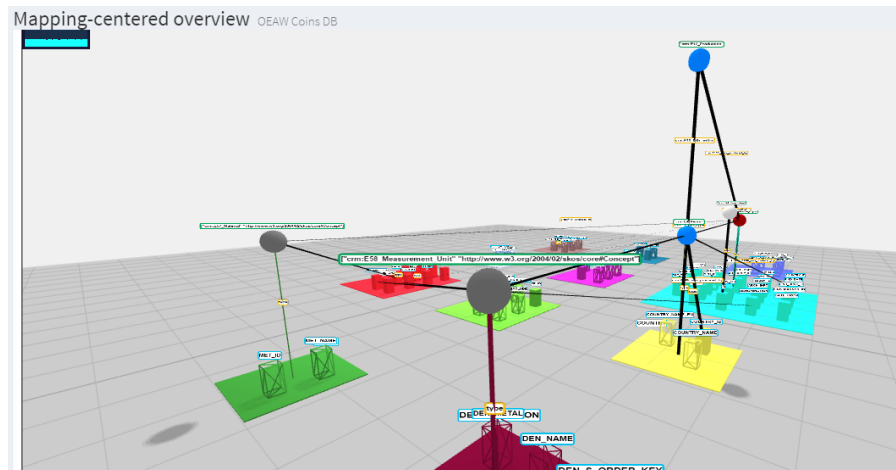


**Figure 33Mapping assertions of source and target schema (2)**

## 4.7 INSTANCE REPRESENTATION

Despite mapping considerations, Instance Representation is a main functionality which depicts the results. You may explore generated instances when the mapping process is completed. It provides an interactive behavior which facilitates users to navigate and explore instances.

At first, you are able to search instances either directly giving their URI or through their class. For example a generated instance of coin can be found by searching "http://www.oeaw.ac.at/COIN/627" or based on its class "http://www.cidoc-crm.org/cidoc-crm/E22_Man-Made_Object".



**Figure 34 Search of instances**

The implementation finds automatically the requested instance and creates an interactive illustrating its related resources. More precisely, you may explore in incoming and outgoing properties between the focused instance and others nodes (i.e. instances, classes and literals) displaying both their labels and URIs. Finally, the graph allows you to navigate through nodes by clicking on their labels. Figure 35 depicts an indicative example of coin. The yellow-colored circle indicates its type; grey-colored triangles are related instances while green-colored squares are literals.
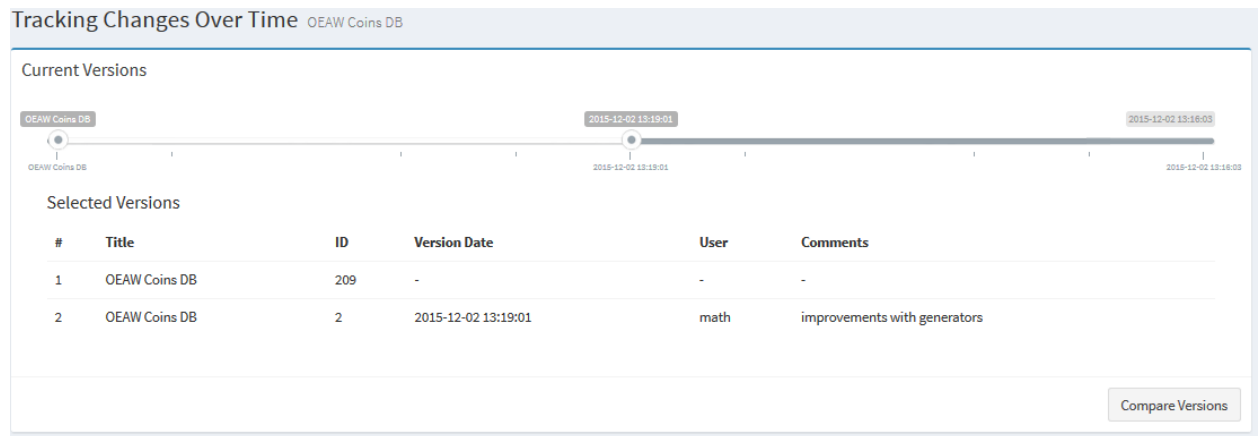
Figure 35 Instance representation

Below you can see a specific focused relation between instance of coin and its class, depicting their URIs (Figure 36).



Figure 36 Focused instance: URI display

## 4.8 TRACKING CHANGES OVER TIME

System provides the capability of monitoring the evolution of a mapping over time. You are able to identify all available versions of specific mapping and compare them. Firstly, a slider illustrates all versions based on date of creation while the current mapping is first in line. Also, it depicts all available information such as title, ID, date, user and comments. You may select two versions and track their changes by clicking **Compare Versions** button.

**Figure 37 Manage Versions of Mapping**

After the required analysis, system tracks the changes about selected versions. It offers a side-by-side projection based on the textual comparison of X3ML files. The yellow color indicates that the rows differ partly, while the red color shows that the mappings differ completely as regards this particular part. An indicative example is depicted in Figure 1
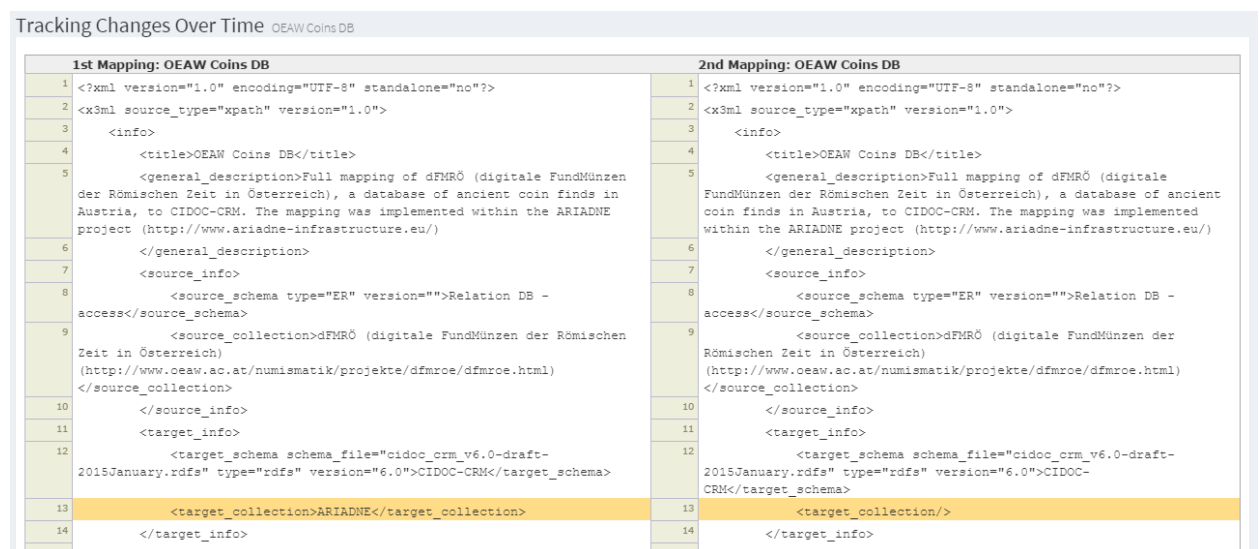


**Figure 38 Comparion of versions**

## 4.9 WARNINGS AND ERRORS

Finally, some additional information can be provided including warnings and errors during the analysis of mappings. The system informs you about any background error describing the reason and related functionality. Any kind of warning or error contains priorities (low, medium and high) to inform user about their importance. We can see the home page's counters in Figure 39.
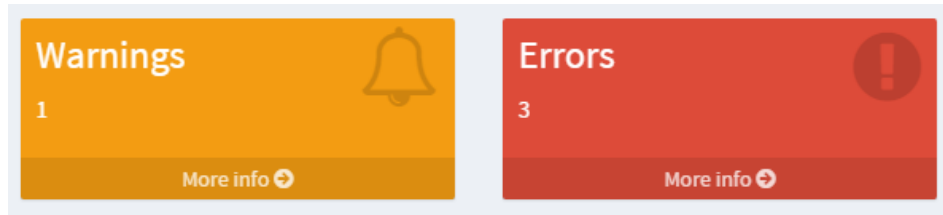
Figure 39 Warnings and errors

Moreover, each mapping is been evaluated in order to check several basic components such as:

- Source schema file
- Target schema file
- Versions of schemas
- Existence of *mapping* elements
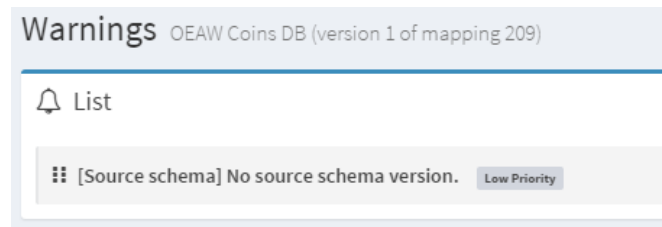- File of generated instances (*.rdfs)
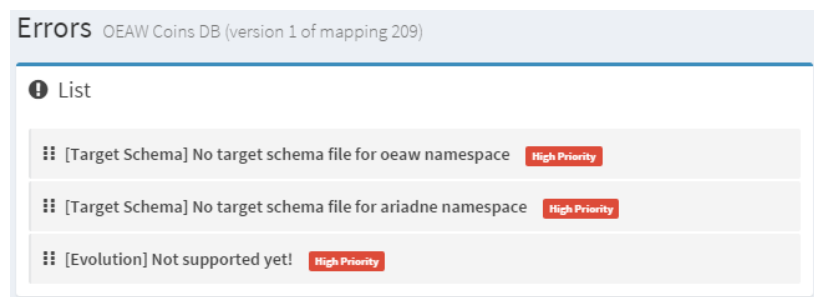


Figure 40 Errors list



Figure 41 Warnings list

# 5 COMPARISON OF MAPPINGS

The other major functionality is the comparison of mappings. It considers two mappings which are compared in the context of their initial form (textual), mapping assertions (graphical) and generated results (instances). You are able to give as input two mapping IDs and system undertakes to retrieve and compare them. The **Home** page (Figure 42) contains a sidebar which provides quick navigation through the available comparison functionalities. The right side of page constitutes the main content containing mappings' information and short description about each provided functionality.
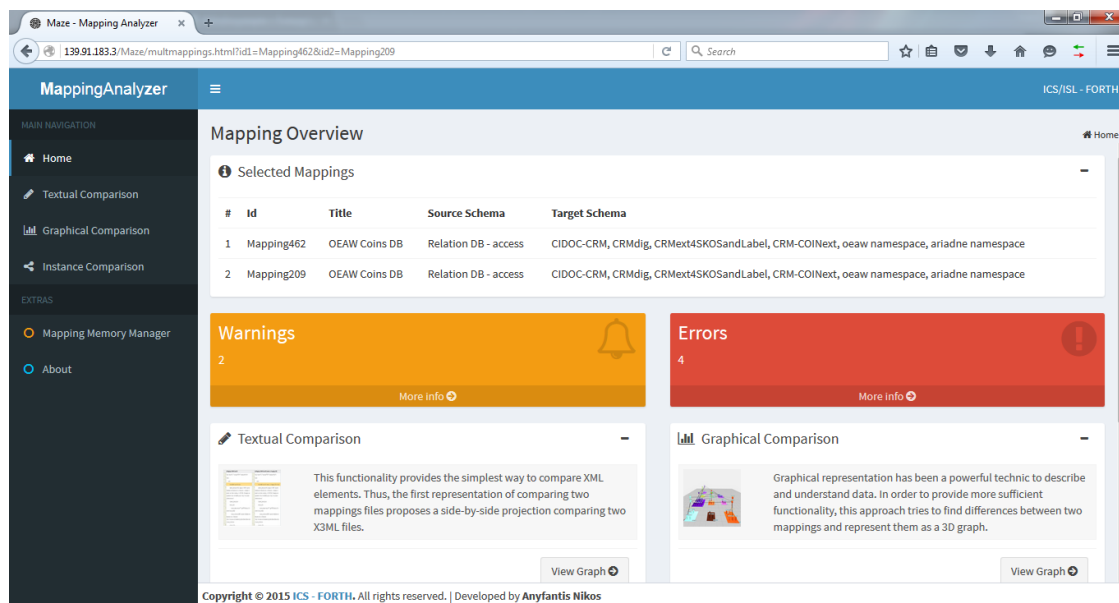
**Figure 42 Home page of comparison**

## 5.1 TEXTUAL COMPARISON

Textual Comparison displays a side-by-side projection based on the X3ML language and its syntactic elements. In this view you are able to identify rows which differ partly or completely. The left side represents the first mapping, while the right side illustrates the second mapping. Each side contains the number of each row in order to help the comparison process.

Also, we mark with different colors the differences. The rows which remain the same have not background color (white). In contrast, differences are colored yellow or red background color depending how they differ. The yellow color indicates that the rows differ partly, while the red color shows that the mappings differ completely as regards this particular part. Figure 43 shows the Textual Comparison of two mappings where there is a partly difference, while Figure 44 depicts a missing element from second mapping.

Figure 43 Partly difference



Figure 44 Missing element

## 5.2 GRAPHICAL COMPARISON

Graphical representation is a powerful technic to describe and understand data. This comparison tries to cover rising issues such as direct comparison of mapping assertions and depict those, combining source and target schemas.

As regards the representation, source schema is depicted at the flour of scene as tree. The structure is started as the center (root node), while nested elements are connected based on its structure. You can select covered elements and explore mappings according to target schema. Also, the source schemas of selected mappings are combined in order to create a common graph. Despite the source schema structure, users are capable to view covered elements directly. If an element is not covered by the mapping, it has transparent substance in scene.

The target schema is depicted at the upper level, displaying differences, related classes and properties. For instance, Figure 45 shows our running example, comparing two mapping about coins. Since you select a specific

element, you may see related classes. The first mapping is presented as sphere, while the second as rhombus. The depicted connections show relations between schemas and properties about target schema. Green labels describe names of classes, while the orange labels are names of properties.
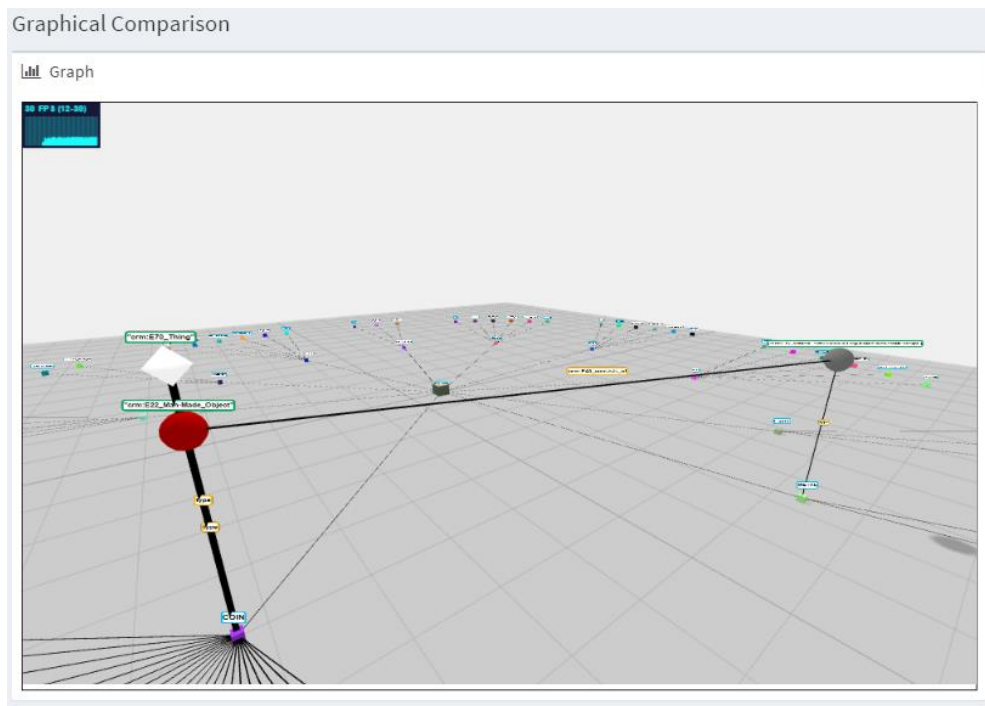


Figure 45 Graphical comparison

More precisely, below you can see more closely the mappings. The white rhombus indicates that first mapping describes coins as "E70_Thing", while red sphere shows that the second approach maps coins as "E22_Man-made_Object". Also, "E22_Man-made_Object" is connected with "E57_Material" through "P45_consists_of" property.
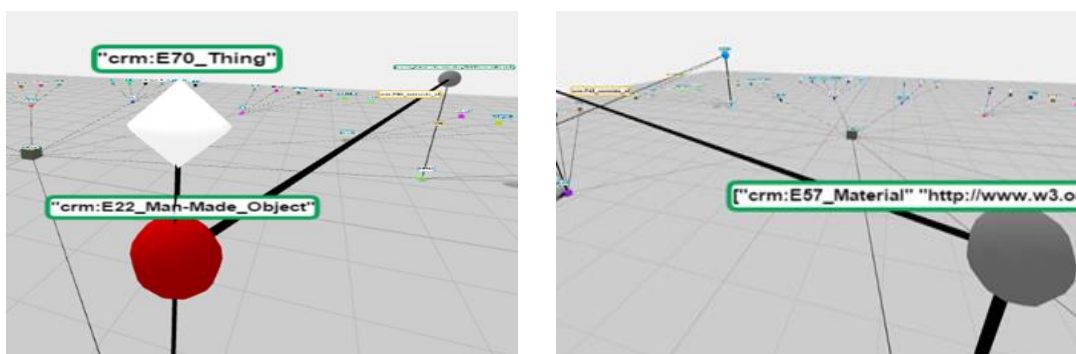


Figure 46 Comparison of specific concepts

## 5.3 INSTANCE COMPARISON

This functionality constitutes a useful method to identify changes according to the generated results. The system compares generated instances between different mappings offering two graphs which display the same concept.

You are able to search instances either based on their URIs or their classes. Figure 47 draws two autocomplete text fields which search automatically concepts. **Search Instance** text field regards direct search on generated URIs, while **Search Instance of Class** regards URIs of available classes. By clicking the search button, we create a graph for selected instance.



**Figure 47 Search of Instances**

Two interactive graphs are available to depict parallel instances. They present all the related resources (instances and classes), incoming and outgoing properties. Also, the graphs offer you exploration technics for navigation purposes between the generated instances. Finally, you may identify differences between them through the side-by-side projection.
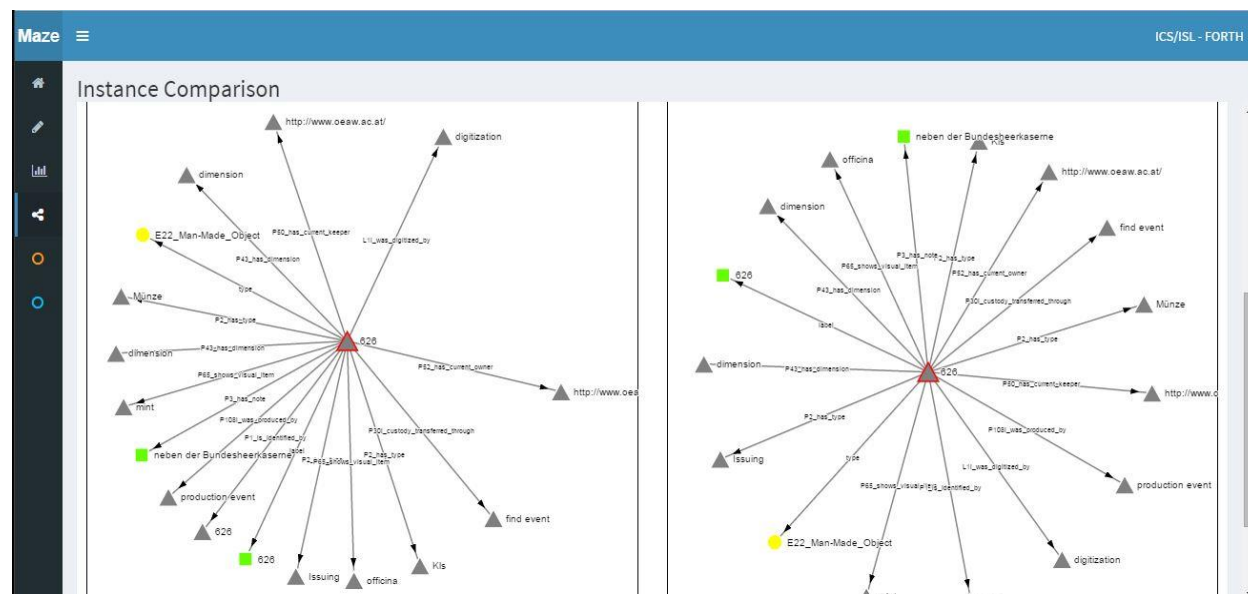


**Figure 48 Side-by-side projection of Instance Comparison**

## 5.4 WARNINGS AND ERRORS

Finally, some additional information can be provided including warnings and errors during the comparison of mappings. The system informs you about any background error describing the reason and related functionality. Any kind of warning or error contains priorities (low, medium and high) to inform user about their importance. We can see the home page's counters in Figure 49.

**Figure 49 Warnings and errors**

Each mapping is been evaluated in order to check several basic components such as:

- Source schema file
- Target schema file
- Versions of schemas
- Existence of *mapping* elements
- File of generated instances (*.rdfs)

# 6 TECHNICAL INFORMATION

## 6.1 GLOBAL ARCHITECTURE

The implementation has been developed as Maven web application project. It has been written mainly in Java and JavaScript, while we have used other web development technologies such as HTML5[4], CSS3[5] and jQuery[6]. The implementation contains about 25000 lines of source code. Maze consists of two main environments (Server and Client), while there are external components which contribute to several functionalities. Figure 50 depicts the Maze global architecture.

**Server.** The server-side part is a Java application. It was been developed in order to provide a complete API of all provided functionalities. It was designed as a REST based-Web Service for X3ML mappings using the Java API for Restful Service (JAX-RS API) with Jersey that exposes multiple resources.

**Client**. The client-side environment consumes the server's services as XML or JSON data and provides representations using all the latest Web features like HTML5, CSS3, Ajax and JQuery that all modern browsers support. Finally, it uses multiple web libraries for visualizing purposes.

**External Components**. The Mapping Memory Manager (3M) and 3M Editor work as intermediary between users and Maze. Finally, several services from 3M Editor constitutes important components, providing X3ML, source, target and other necessary files in order to analyze them.

Maze components are further explained in more detail in the next subsections.
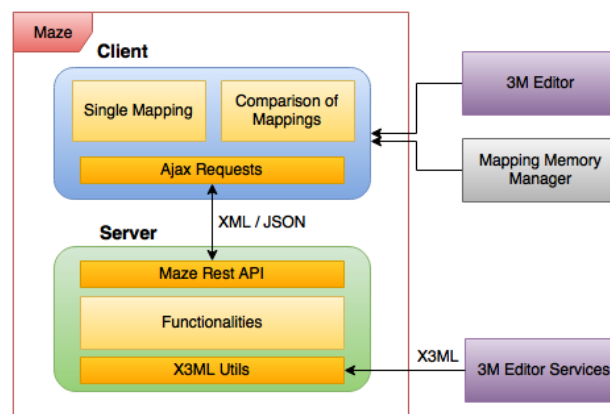


**Figure 50 Global Architecture**

---

[4] A markup language used for content on the WWW. https://en.wikipedia.org/wiki/HTML5
[5] The latest (Feb 2016) standard for CSS. http://www.w3schools.com/css/css3_intro.asp
[6] jQuery a fast and feature-rich JavaScript library. https://jquery.com/

## 6.2 SERVER-SIDE ENVIRONMENT

Considering the backend of our tool, we have designed an effective system which undertakes to analyze mappings and provide generated results as services. It is written in Java for several reasons. First, this is the language we master the best. Second, Java is world-wide used and possesses a large community of developer which facilitate the access to information and code examples. Finally, the main libraries that are used in this implementation (Jena, Jersey, etc.) were existing in Java.

The server-side environment core consists of six major components: Maze Rest API, Controller, Provided Functionalities, Data Generators, Utilities and Services Consumers. In Figure 51 the overall structure of server is depicted.
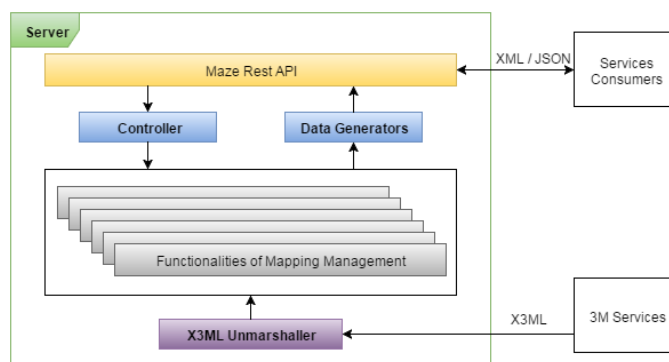


**Figure 51 Server-side Environment**

The Maze Rest API service is responsible for exchanging data between the server and services consumers. The REST based API is provided in order to serve clients for X3ML mappings. It consumes and produces XML or JSON format depending on the requested data forms.

The Controller is the orchestration component of the server's architecture. It is aware of any operation that takes place in the backend. In more details, the Controller is responsible for making decisions for every process workflow regarding requested operations, and controlling the collaboration of Maze subcomponents.

The Provided Functionalities implements the main processes regarding each request. They consists of five packages, containing data structures, functions, etc. These packages regard source and target reasoners, coverage metrics, instance and mapping rules analyzer.

The Data Generators' role is twofold: (a) they are responsible for conducting the implemented functions, ordering the involved executions and (b) they generate data in order to be produced by services.

The Utilities facilitate several functionalities, including general procedures. They are an intermediate component which receives documents from 3M Editor, load ontology models to memory and manipulates X3ML files (i.e. marshaling, unmarshaling XML files). Moreover, they formulate dynamic SparQL queries in order to serve particular operations.

The Services Consumers constitutes an external component which consumes server's results. Although our web application plays the role of consumer, there is the capability of retrieving data from other external clients.

Besides these basic components, the server use a configuration file, provided by a java file that contains all the information required by the tool (Resources.java). Concluding, we support log files for better monitoring of tool which are also accessible by service.

## 6.3 CLIENT-SIDE ENVIRONMENT

The client-side environment constitutes the user interface of our system. It has been implemented based on modern design patterns, following the basic user interface design principles. As regards development information, it is written in JavaScript and jQuery, using the latest web technologies such as HTML5, CSS3, Ajax requests, etc. Finally, it has been designed in order to be adapted to mobile devices and to be supported by the majority of modern web browsers such as Google Chrome, Mozilla Firefox, Opera, etc.
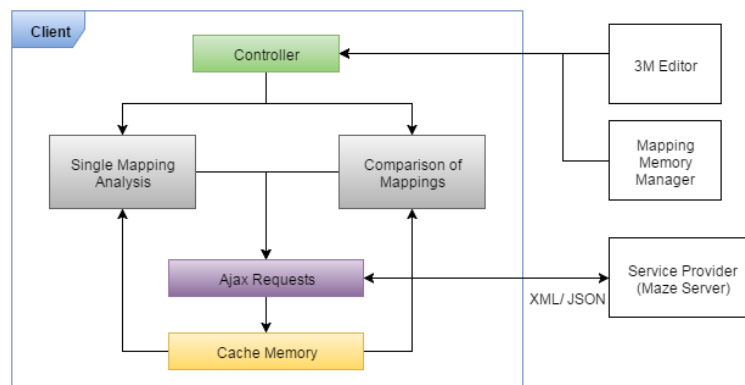


**Figure 52 Client-side Environment**

As regards the client's architecture (Figure 52), it consists of five major components: Controller, Single Mapping Analysis and Comparison of Mappings pages, Ajax Requests and Cache Memory manager. The Controller constitutes the administration component which manages internal processes. It is responsible for addressing users' needs to the corresponded components and instructing the exchanged data. The Single Mapping Analysis page provides the majority of implemented functionalities. It regards a specific mapping, providing comprehensive view in order to visualize analyzed results such as coverage metrics, source and target schema graphs instance representation, etc. The Comparison of Mappings page takes as input two mappings and is responsible for depicting comparison functionalities. The Ajax Requests works as intermediary between client and server, orchestrating data exchanges. It directs the data in order to send and receive the appropriate contents, making asynchronous requests to server's API. The last main component is the Cache Memory manager which stores and retrieves temporary data in session storage of web browsers.

Furthermore, there are external complementary components. The 3M editor and Mapping Memory manager redirect users to our system, while they supply client with required documents and information.

For visualization purposes, we used multiple web-based libraries. We used Bootstrap[7] in order to make front-end web development faster and easier and offer mobile-friendly user interface. Moreover, we used wide-spread web libraries for representation purposes. The most important are: D3[8], Threejs[9], Arborjs[10], Jointjs[11], Chartjs[12], Difflibjs[13], etc. Finally, configuration file is available which allow users to adjust client and gathers the several necessary information, such as location and port of server, url of Rest services, etc.



Figure 53 Web technologies

---

[7] http://getbootstrap.com/
[8] http://d3js.org/
[9] http://threejs.org/
[10] http://arborjs.org/
[11] http://www.jointjs.com/
[12] http://www.chartjs.org/
[13] https://github.com/cemerick/jsdifflib