# WebMIPMap

## Overview

WebMIPMap is a web application specifically tailored for the needs of the Human Brain Project that offers schema mapping utilities. It provides an easy to use web interface where correspondences between schemata can be defined by simply drawing arrows between two tree-form representations and generates declarative representations, under the formalism of TGD rules[[1]](#footnote-1).

Its purpose is allowing registered users of the HBP to create mappings between the MIP schema and hospital or research data schemata. These mappings can be used to translate hospital and research data schema terms to terms of the MIP schema in case this terminology does not exist in the MIP schema. Additionally, WebMIPMap can be used to map the MIP schema to existing ontologies, as well as a web interface to create mapping tasks that can later be downloaded and run on the desktop version (MIPMap).

WebMIPMap was built as an extension of the desktop application “MIPMap”, a schema mapping and data exchange tool also developed by Athens University of Economics and Business for the needs of HBP, although the desktop version focuses equally on the generation of the rules that define the mappings between a source and a target schema as well as on the exchange of data instances between them.

The application offers the option to express a mapping between a source and a target schema through graphically defining the correspondences between them, as well as join conditions, constraints and functional transformations. The tool supports the creation of n:m correspondences allowing complex transformation functions and the assignment of constant values or function generated values. Selection conditions can also be applied on the tables of the source schema.

## Input

WebMIPMap accepts different types of input that can define a relational schema. CSV files, XML Schema Definition (XSD) files and SQL scripts can be loaded as well as relational schemata through connections with supported databases (MySQL/Postgres/Apache Derby). The option to choose from a set of pre-defined schemata stored at the server is also available.

## Output

The final output of the application is a set of well-defined formalized rules that correspond to the schema mappings and options defined be the user. This set of TGD rules is presented to the user through the user interface, while he/she is also given the option to export them in a file to his/her local machine.

## Basic structure – Design patterns

WebMIPMap is a java web application that has been developed in Java (Java 8 release) and JavaScript. The application follows the model-view-controller (MVC) architectural pattern, isolating the application logic from the user interface layer. The View part of WebMIPMap is built solely on JavaScript with the use of jQuery libraries. Its communication with the server is implemented via AJAX requests to Java Servlets. Controller and Model parts of the application are developed in Java.

The client side of the application was developed using HTML along with the JavaScript scripting language. JavaScript was chosen because of its versatility, allowing the creation of a dynamic web page with customizable visual effects as well as client-side behavior of the page. As far as the grapchical representation was concerned, a designing decision was made to make the user interface of WebMIPMap as close as possible to its desktop counterpart. The use of JavaScript allowed the design of an interface that resembles the original, while adding more smoothness in motions and improving ease of use. JavaScript was also chosen for its high performance, being able to process user input and actions and do calculations fast on the user’s processor thus saving bandwidth and strain on the web server.

Instead of raw JavaScript programming the jQuery open source library was used due to its flexibility and cross-browser support. The following plugins and components built on top of jQuery library, were used: jQuery UI[[2]](#footnote-2) for the creation of tabs, panes and dialogs, jQuery Steps[[3]](#footnote-3) for the creation of the wizard dialogs, along with jQuery Validation[[4]](#footnote-4) for preventing step changing or submission on wrong user input, the Rangy Inputs plug-in[[5]](#footnote-5) for manipulation within text areas, jsTree[[6]](#footnote-6) for building interactive trees and jsPlumb[[7]](#footnote-7) for the creation of visual connections between elements.

User actions and their parameters are sent to the server over AJAX HTTP requests and are received by Java Servlet objects. There are two Servlets that handle file uploads during the creation of a new mapping task, one for the source files and another one for the target files accordingly. A single Servlet receives all other requests and sends back the appropriate reply having completed the necessary operations on the server.

A temporary directory is used on the server to store the current session mapping tasks and respective input and output files. In case the user wishes to store this information the mapping task and the files are moved to a unique directory, distinguished by the user id, that is assigned to each registered user of the HBP. Otherwise, when the session is over the mapping task with the respective files are removed from the temporary directory. The SourceFileHandlerServlet is responsible for uploading and removing user files that are part of a new mapping task. A request to this class is made whenever the user selects to add a file to the new mapping scenario. The file is sent to the server and is uploaded to a temporary directory. Regarding files that contain the information on relational schemata and not their data, a maximum limit of 20 MB per file is allowed. For security reasons and space limitations only the first line of the file, enough to describe the schema, is sent during CSV file uploading. This file handling is executed on the client. Removal of the specific file is handled by the same Servlet when it is selected by the user during a new mapping task creation. The Servlet TargetFileHandlerServlet is responsible for the respective actions for the target schema files.

The MappingServlet accepts all remaining requests and calls the appropriate objects and methods. A JSON file is sent with the HTTP response containing all necessary information. If an exception is caught during execution the message is sent over to the client to inform the user on the error. The supported user actions that the Servlet handles are the following:

• Load input

1. Define source and target schema files (CSV, XSD, SQL) or connection parameters to a database (MySQL/Postgres/Apache Derby) for a new mapping task.
2. Load a saved mapping task in xml format.

• Create a correspondence

1. Create a 1:1 correspondence between two elements.
2. Creating more than one correspondences in the same target element results in the creation of a n:1 correspondence.
3. Provide a constant value for a target element.
4. Create a transformation function between elements.
   1. Supported transformations are numeric, string, casting and logical.
   2. Transformation functions can take as input more than one source elements.
5. Define a join condition or foreign key between two elements.
6. Define a selection condition.

• Generate TGD rules.

• Requesting SQL script for external data exchange process.

• Get output

1. Export generated TGD rules.
2. Save a mapping task.

• Delete a saved mapping task.

The core of WebMIPMap is the ‘mipmapEngine’ module, being the component responsible for creating the schema Node objects and the connections between them and generating the TGD rules according to them. The module also produces the SQL scripts that can be used for external data exchange process. It is used as an external library and is the same module as the one used by the counterpart desktop application, built as a reusable component. The MIPMap project has been created as a module suite with its components and external libraries each being a separate module, while of them are built together to form the whole project.

WebMIPMap can take as input:

* A set of csv files forming a database schema with each one corresponding to a table. For csv input the OpenCsv[[8]](#footnote-8) external library is used.
* An SQL script containing the schema definition, taking into consideration data types and table constraints and references. The SQL script is parsed using the JSqlParser and is translated to a traversable hierarchy of Java classes. MySQL, PostgreSQL and Oracle syntax is supported.
* An XSD file containing schema definition in a hierarchy of elements containing table constraints and references.
* A relational database (MySQL/Postgres/Apache Derby) schema itself, given the connection uri to it and credentials. For this kind of input the Jdbc API is used, respective to each database. Jdbc implements the protocol for transferring the query and result between the application and the database.
* Ready, pre-saved relational schemata, which are stored in SQL files.

All mentioned libraries have been added to the project as modules. When the input data are loaded a set of nodes (INodes) is built, structured in a tree form, for the schema representation. The composition of the INode trees is materialized a set of DAO (Data Access Object) classes. This way an abstract layer is interposed between the interface calls and the persistence layer adding more flexibility as to what database backend can be used by hiding the database details from the operations.

The creation of the INode and Correspondence classes and the ones that represent operations on them follows the Prototype design pattern; while behaviorally the Visitor pattern is mainly applied allowing different kind of operations to be performed on the elements based on both their visitor’s and their own type.

## Tuple-generating dependencies (TGDs) generation process

The application supports simple 1:1 correspondences between elements, as well as n:1 correspondences with complex transformation functions and the assignment of constant values or function generated values. Selection conditions can also be applied.

If a transformation function has been applied on the source data, firstly the syntax of the function expression is evaluated by the JEP Java library. The JEP library has been extended with the addition of new functions (Length, Datetime, ToDouble, ToString, ToLowerCase etc) or expansion of existing ones; the “substring” function, for example, can take as optional input the end index as well.

The concept behind the rewriting process can be summarized as follows: producing the “optimal” solution during the data exchange procedure translates into discarding the redundant produced tuples while at the same time merging tuples that correspond to the same instance. One way to achieve this could have been to generate the so-called canonical solution and then apply a post-processing algorithm on all produced tuples. This approach however requires very high computing time and can hardly scale to large datasets. Instead, possible redundant target tuples, called homomorphisms, are recognized already in the TGD level and since the number of TGDs is considerably smaller than the number of instances, the computing procedure is much more efficient.

Taking into account the provided correspondences, constant assignments and the constraints declared, the original algorithm of the Clio[[9]](#footnote-9) system is used to generate the candidate TGDs. Next, the rewriting algorithm recognizes possible cases of homomorphism among these rules, either in the form of subsumptions or coverages. These constraints may have been expressed by the user as foreign key relationships or joins between columns of either the source or target schema. After potential generation of redundant tuples has been detected the candidate TGDs are rewritten to the optimal ones that produce the optimal results. These computations and changes are expressed in the generated SQL executable in the form of negations and joins. Self-joins, applied by relations between duplications of the same table, can also be taken into account using the same rewriting algorithm to produce the core solution. Lastly, primary key constraints in the target schema and functional dependencies between elements, often expressed as EGDs, can be selected to be taken into consideration when computing the final solution.

## User Authentication

User authentication in WebMIPMap is achieved through the OpenID OAuth2 authentication protocol. WebMIPMap has been designed as a Spring application and uses Spring Security to manage the OpenID authentication.

The application has been registered to the Human Brain Project OpenID provider. When a request for a resource is made by a non-authenticated user, the user is redirected to the Human Brain Project login page. After providing his/her credentials, the Human Brain OpenID provider checks the validity of the provided information and an OAuth2 token is issued. The authenticated user is then redirected back to the WebMIPMap application main page and the OAuth2 token is sent with every user request.

When an authenticated with the Human Brain Project user logs in for the first time to the WebMIPMap application, a folder is created on the server, on the path specified by the administrator on the configuration file. The folder is dedicated to the specific user and is named after the user’s full name, as it has been declared on the Human Brain Project user details (taken by the application as “name” from the java security Principal object). The folder is used for storing mapping tasks created by the user, along with the needed files that have been uploaded by the user, and accessing previous saved tasks. Each user has access only to his/her mapping tasks.

A user log out option is also available that destroys the generated authentication token allowing no more access to the WebMIPMap resources until a future user login.

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995). [↑](#footnote-ref-1)
2. <https://jqueryui.com/> [↑](#footnote-ref-2)
3. <http://www.jquery-steps.com/> [↑](#footnote-ref-3)
4. <http://jqueryvalidation.org/> [↑](#footnote-ref-4)
5. <https://plugins.jquery.com/rangyinputs/> [↑](#footnote-ref-5)
6. <http://www.jstree.com/> [↑](#footnote-ref-6)
7. <https://jsplumbtoolkit.com/> [↑](#footnote-ref-7)
8. <http://opencsv.sourceforge.net> [↑](#footnote-ref-8)
9. L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, R. Fagin - Translating Web Data. In Proc. Of VLDB, pages 598–609, 2002. [↑](#footnote-ref-9)