Project number: 257872

METAFORA

Learning to learn together:

A visual language for social orchestration of educational activities

STREP

ICT / Technology-enhanced learning

D4.2 – Technical report and user manual (Final System)

Due date: 30.8.2013

Submission date: 30.08.2013

Project start: July 1st, 2010 Duration: 38 months

Organization name of lead contractor for this deliverable: KUEI

Project co-funded by the European Commission within the 7th Framework Program (2007-2013)

Dissemination Level

PU Public	**X**
PP Restricted to other programme participants (including the Commission Services)	
RE Restricted to a group specified by the consortium (including the Commission Services)	
CO Confidential, only for members of the consortium (including the Commission Services)	

Document number:	D4.2
Document Title:	Technical report and user manual (Final System)
Work package:	WP4 – Framework System, Domain Tools, and Integration
Document status:	Final
Date:	30.08.2013
Authors:	Thomas Irgang (KUEI), Norbert Sattes (KUEI), Kerstin Pfahler (KUEI), Andreas Lingnau (KUEI), Andreas Harrer (KUEI)

Abstract

In this document we present the final system of the Metafora platform, with the enhanced development and integration of the tools in the Metafora system.
Firstly the design decisions and architecture of the technical system will be provided. Further the tools of the Metafora platform with its technologies will be presented, followed by an introduction to user acceptance and evaluation. Finally a summary of the proliferation strategy for open source, open access and how to integrate external tools into the framework will be explained.

Keyword list

Metafora platform, web-based, Google Web Toolkit, XML, Internationalization, XMPP, user interface, visual language, system architecture

DOCUMENT CHANGE LOG

Document Issue.	Date	Reasons for change
Early draft	23.07.2013	First draft
Draft	21.08.2013	Internal changes requested
Draft	29.08.2013	External changes requested (M. Mavrikis, Quality Manager)
Final version	30.08.2013	

APPLICABLE AND REFERENCE DOCUMENTS (A/R)

A/R	Reference	Title
1	Annex I to the Grant Agreement	Description of Work (DoW)
2	Consortium Agreement	Consortium Agreement
3	Grant Agreement	Grant Agreement
4	D1.2	Requirements analysis – First report
5	D2.1	Report on Visual Language for Learning Processes
6	D4.1	Technical report on the framework system (Prototype 1)
7	D4.3	Metafora Microworlds
8	D5.2	Diagnosis abstraction layer document
9	D6.2	Requirements analysis – First report

Table of Contents

List of Illustrations

Abbreviations used

GWT – Google Web Toolkit

AJAX – Asynchronous JavaScript and XML

XML – Extensible Markup Language

XMPP – Extensible Messaging and Presence Protocol

AI – Artificial Intelligence

UI – User Interface

GUI – Graphical User Interface

1. Executive Summary

This Deliverable describes the final system of the Metafora project. It is a report on the technical framework system and the integration of all applications, used in the Metafora project, such as several microworlds, a tool for planning and the discussion environment. Additionally, it will present a tool for designing the visual language of Metafora according to the needs of a learning scenario and a tool for reflection. A major focus of this report is the data management, the flexibility of the system and design decisions, implemented interfaces, as well as awareness features and functionality.

The first part of this document will be dedicated to the design decisions of the technical Metafora system. It will present the different modes of awareness, the architecture and security of the framework system and summarize the character of flexibility and openness of the system.

Afterwards the deliverable will present the tools of the framework system. The main embedding tool called *Home*, the *Planning Tool* which integrates the visual language, a tool for reflection, followed by an application for designing the visual language and finalized by the *Workbench* of Metafora.

Subsequently section 4 will present the MeET US study with the purpose to evaluate the user acceptance of the system.

Deliverable 4.2 will be concluded with the open source and open access concept of Metafora which was constructed in cooperation with WP7 and ends with a manual on how to integrate external tools into the Metafora framework system.

In the attachment you will find the **User Manual** of the Metafora system, which explains how to use all the features and functionalities of the system from the target audience's point of view.

2. Design Principles

In this section we will describe our design decisions and principles. We will explain the modes of awareness of the Metafora system, the designed platform architecture with its security measures and end the section with the intended flexibility and openness of the system.

2.1. Modes of awareness

Working in Metafora, learners are going to use a variety of learning tools, each of them having its own focus and context and thus learners might lose track of their current task and of what their fellows are doing. To support awareness for these factors, Metafora's analysis components create information, that is adequately displayed and such help the learners keep track of what is happening “nearby”.
Across the Metafora platform, almost all Tools and components support awareness in some form. Nevertheless, a few of them have a special focus on this topic. These will be presented in detail in later chapters, but are in particular:

- The *Chat*: a communication component built into the integrating module *Home* and supporting Referable Objects[1]
- The *Notification Area*: an area for displaying Notifications
- The *Reflection Tool*: a Tool to support reflection on the learning process and process awareness
- The *Workbench*: a central overview component providing the user with information about document storage and different kinds of awareness

2.1.1. Categories of awareness

To define what information is relevant to the learner and how it should be visualized, we used two categorizations for awareness information, the first covering the type of

[1] See Deliverable 4.3

awareness provided, the second one its visualization / representation [5]. The first categorization distinguishes three kinds of awareness-information:

- *content awareness*: What interesting things are there? This category of information is important when learners want to know, what artefacts have been created and manipulated.
- *social awareness*: Who is logged in and working on which part of the Challenge? This is important, when users work remotely and would like to collaborate with others.
- *process awareness*: When did something happen? This kind of information is important while working on complex processes, but especially when it comes to reflecting about what happened throughout these processes.

The second system of categorization, that helped to decide, what components of Metafora display what information, is based on [6]. It features the two dimensions:

- *synchronous vs. asynchronous*: These can clearly be distinguished by considering the online status of users. The synchronous case mostly prevails when users work in class at the same time, while the asynchronous one gains importance after school, when the learner wants to know, what has been done and what events have occurred, while he was offline.
- *coupled vs. uncoupled*: This dimension defines a scope of interest or context, about which the user should be more or less intensely informed. E.g. events generated in other groups concern the user less than events that are related to his current task and happen within his group.

For the coupled-dimension we found two ranges, which both are reasonable in certain situations: On the one hand coupled might mean information generated in the same challenge, the user is working on. This information might be related to his current task and therefore be relevant regardless of where it comes from. On the other hand, coupled might mean information within the users group such as information related to social awareness. As a compromise we defined coupled awareness information to be generated in the same group as the user is logged in to,

and in the same challenge, he is working on, but subdivided uncoupled by information from the same challenge. The following table, adapting and extending [6], highlights all aspects of this categorization:

	same challenge		Other challenge
	coupled	Uncoupled	
synchronous	What is happening in the context of the current task?	What is happening in this Challenge?	What is happening in other Challenges?
asynchronous	What has happened in the context of the current task?	What has happened in this Challenge?	What has happened in other Challenges?
	same group	Other group	

Table 1: Modes of awareness

Both systems of categorization were useful for the design of awareness features in Metafora, but especially for clarifying the differences between and focus of the Reflection Tool and the Workbench and their individual modes of presentation.

2.1.2. Awareness features

Support for awareness on the Metafora platform already starts during the login-process: previously selected login-information is displayed during each step of the process, facilitating the user's orientation. Afterwards the integrating component of Metafora, called *Home*, displays social awareness information, in particular the online status of other group members and the selected login-data, in its *Group Info* tab, available at any time, the user is logged in to Metafora.

In the context of awareness-support, the Workbench's *Breaking News* section and the Reflection Tool gain special meaning: both are capable of displaying information, that is marked to be relevant, i.e. landmarks, but they have different focuses and purposes and thus show their information differently. The *Breaking News* aims at informing the user, about what is happening and what has happened recently (we

defined this to be synchronous information) within Metafora in compact fashion. By also showing landmarks from other groups, it creates a “gamification”-effect that invites one group to compare itself to another, but also helps to identify fellow learners, who could help at a certain task. Due to our fluid group concept as described later on in chapter 2.2, we decided to let the *Breaking News* show landmarks of other groups in the same challenge from across all challenges, since a learner might want to have an overview of peers’ actions, when a certain landmark is triggered there. So the *Breaking News* covers part of the synchronous-row in the modes of awareness-table above.

In contrast to the Breaking News, the *Reflection Tool's* focus [2] is to support the learners' reflection about their whole learning process, as well as to enable them to look up, what happened during their absence. Whereas the Breaking News has a very close-to-now focus, the *Reflection Tool* covers the whole learning process of a group within a challenge and thus the whole coupled-column in the modes of awareness-table.

2.1.3. Chat, Feedback and Help

Metafora integrates three more types of information, which require special awareness mechanisms, since their aim is not only to passively provide the user with information when he needs it, as do landmarks in Workbench and Reflection Tool, but also to induce a reaction from the user. These are Feedback-, Help- and Chat-Messages. Feedback Messages will be covered in the chapters 2.2 and 4.1. These Messages are displayed such that they more or less interrupt the addressed user in his workflow, catching his attention, and can be invoked directly by users via the Feedback Tool, that is reachable through the Workbench and the *Tools* section of the Home. Chat messages have been decided to behave like non-interruptive Feedback-Messages, since we expected them in high numbers and Help-Messages like high-interruptive ones, due to their importance and expected rarity.

2.2. Architecture

Metafora was designed as a web-based learning platform with the focus on the *L2L2*[2] concepts and support of the learning to learn together principle. The core feature of Metafora is to support students in collaboratively planning their activities in a learning scenario. It was designed as a platform which integrates web-based, domain-specific learning microworlds. Our Metafora design concept is focused on the one hand on easy integration of existing microworlds and extension of the developed platform and on the other hand on strong modularization to allow easy reuse of the developed components.

Our proposal for Metafora's architecture can be seen as a modified *blackboard architecture* [1]: several learning tools and analytic components can be used at the same time and do not interact directly with each other (loose coupling). All analytic components are able to read and take up results from other analytic components and all learning tools are able to send commands for interoperation between the tools, because a unified language is used between all these components. This allows a flexible combination of learning tools and associated domain-specific and tool-specific analytic components with cross-tool or domain-independent components and can also be used with already existing indicator-based analyses, as e.g. from MiGen's eXpresser. An example of tool interaction is contained in [2] and will be presented at CRIWG 2013[3] conference.

[2] L2L2: Learning to learn together

[3] http://criwg2013.vuw.ac.nz/

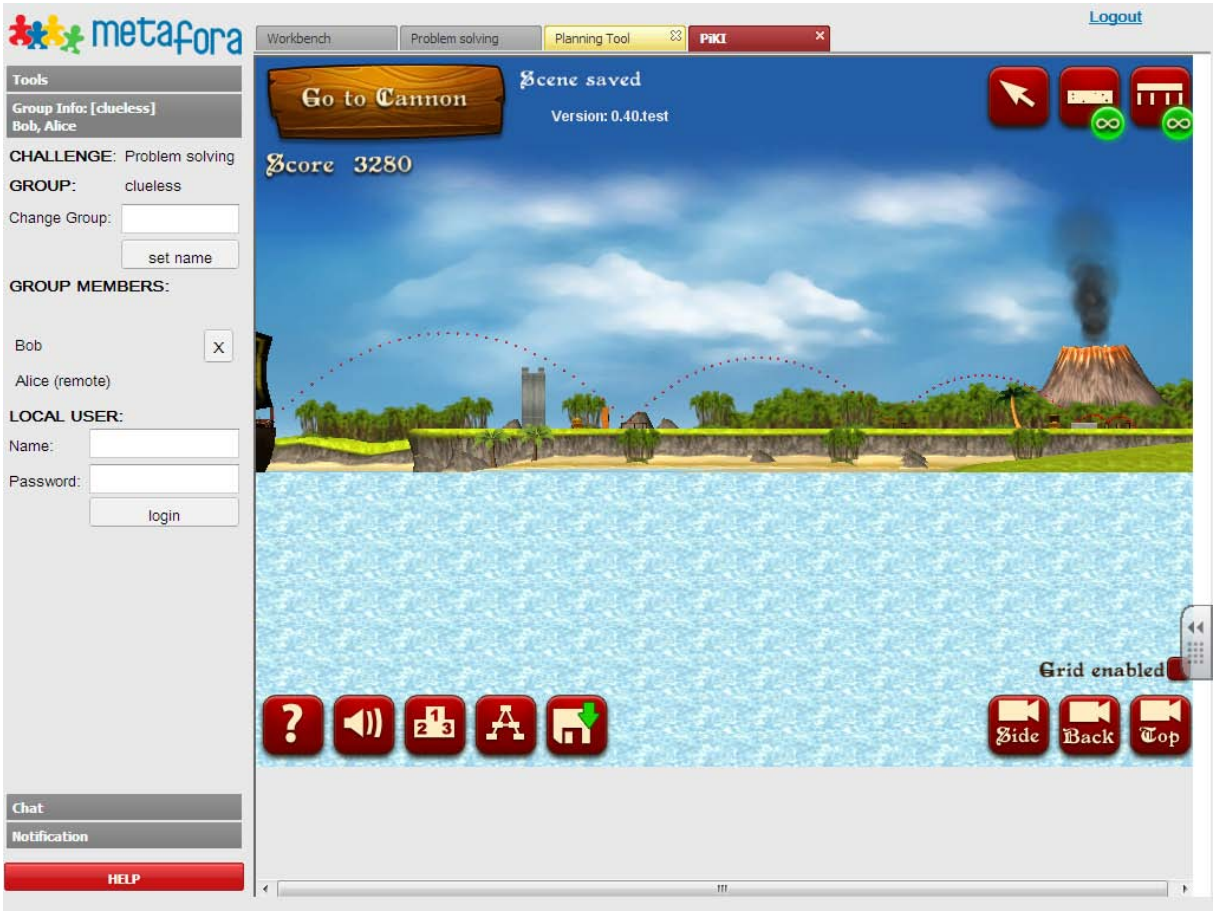

Image 1: Metafora system with the physics microworld PiKI, visible in the center part of the screen.

Image 1 shows a screenshot of the Metafora user interface after the login. The Metafora platform interface with the sections *Tools*, *Group Info*, *Chat*, *Notification* and the help button is visible on the left side. The Tools panel contains direct links to all integrated tools like PiKI[4] or LASAD[5]. This can be used for the quick access to the tools, but the states of the tools cannot be stored or opened again. For the usage of the tools in a learning scenario, they can be started with the help of the Planning Tool, which allows reentering the last saved state in the microworld. The *Group Info* panel, which is shown in Image 1, shows information about the current challenge and group. Students can also change their group with this panel without logging out and in again. With the help of this panel it is also possible to login more than one student at one computer and it shows which users are remote or local in the group. The chat

[4] Pirates of the Kinematic Island (PiKI) is a microworld, developed in context of Metafora from Testaluna.

[5] LASAD argumentation tool, accessible at http://cscwlab.in.tu-clausthal.de/lasad/

panel contains a tool independent chat for all users in the same group and challenge. The notifications panel collects all old feedback messages of the student's session.

The tab bar above the main space in Image 1 show the open tools as tabs. The selected tool is PiKI which is shown in the center part of the screen and the other open tabs are Planning Tool, Workbench and the challenge description, which is named with the challenge name *Problem solving*. A logout button is shown in the upper right corner. This button is visible in all Metafora interfaces coherently at this position.

The Metafora platform integrates a few core components, some support modules and a set of learning components. The Metafora core components are the embedding tool which is called "Home", the Planning Tool, Workbench and Reflection Tool. The support modules are Metafora Service Module, PlaTO, Metusalem, Planning Tool Map Creator, LASAD Map Creator, XMPP Bridge and Messaging Tool. Metafora already supports the microworlds PiKI, Sus-X, eXpresser, Juggler, 3D-Math and the discussion environment LASAD.

The most central L2L2 component of Metafora is the Planning Tool. With the help of the Planning Tool the students collaboratively plan their activities for a challenge and refine their plan. If they have finished planning they can use their plan to enter the referred mircoworlds. Microworlds are referred to with the help of resource cards. Those cards are linked with the help of a default URL to a microworld. If a student "starts using" a resource card, the Planning Tool completes this URL with plan information and sends a XML action to the command channel (cf. section 2.2). The Home application reacts to this action and opens the microworld as new tab on the client of the student. With the help of the plan information the microworld can update the tool URL of the resource card to a new one, which links to the current microworld instance of the student. This allows the student and other students to reenter the same microworld instance in a later session. The state of the tool can be stored persistently by the tool itself, like it is done by eXpresser, or in the Metafora document store, like it is done in PiKI. The Metafora Service Module provides a servlet which allows tools to upload documents to the Metafora document store with

the help of an HTTP post request and receive documents with an HTTP get request. The document id, which is required for downloading of documents can be requested with the help of a XML command. The Planning Tool and LASAD use named workspaces. To support linking of resource cards to existing named workspaces we included a Planning Tool Map Creator and a LASAD Map Creator, which allows students to create new workspaces or connect existing workspaces with a resource card. In case of Planning Tool this feature allows students to use it like a grapical hypertext (a la Wikipedia), and refine steps of their plan hierarchically in other plans.

The inter-tool communication in Metafora is done with an extension of the XMPP protocol. The extension XEP-0045: multi user chat[6] follows the IRC idea with support of persistent and non persistent chat rooms and common user roles like moderator or admin. An advantage of this extension is that all common XMPP servers support creation of non persistent and persistent multi user chats (MUCs) without special configuration and most of the open XMPP servers like Jabber.org[7] support creation of MUCs without special rights. Another advantage of XMPP and especially this extension is that most of the libraries, which are available for all common programming languages, support multi user chats. Metafora uses three different XMPP multi user chats to group the inter-tool communication. Every integrated component should post log messages for user interaction to the logger multi user chat. High level interaction feedback and results of analysis components like indicators and landmarks[8] should be posted to the analysis channel. For inter-tool commands like opening of microworlds Metafora uses the command channel. Image 2 shows a simplified view of the Metafora architecture.

[6] http://xmpp.org/extensions/xep-0045.html

[7] http://www.jabber.org/

[8] Landmarks are important steps in the students learning process, like finishing of an activity.

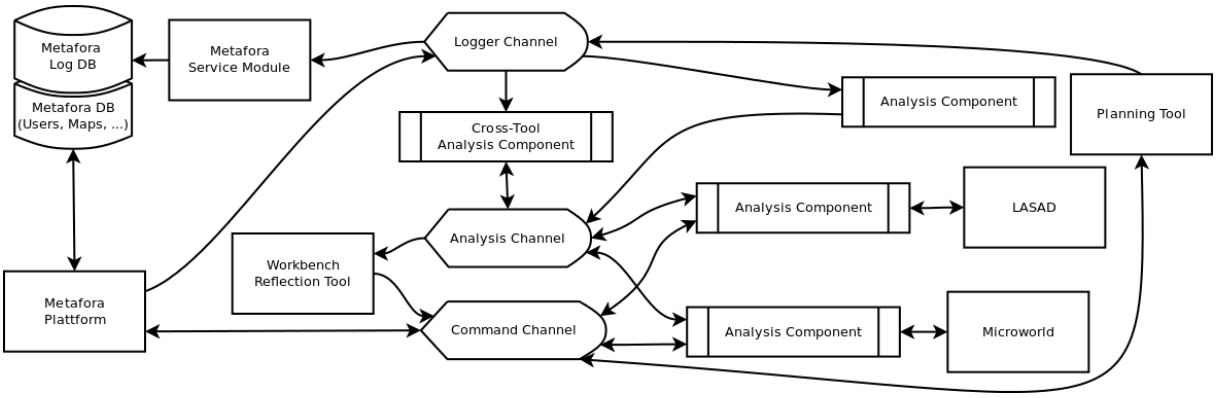

Image 2: Architecture overview of the Metafora system.

The Metafora architecture allows the integration of tools with very different technological basis. Server based tools LASAD and Planning Tool use GWT and the also server based microworld eXpresser uses the GWT App Engine, which allows to deploy the tool on Google servers. The microworlds PiKI, Juggler and 3D-Math are implemented with the Unity Framework, which is client side only, like Flash games. The integrated Sus-X mircoworld is a Java application with Java web start support. For the automated login of local students into the used tool and for tool interaction the tools require user information and need to be able to identify the client. Because of the support of multiple users on a client and the support of flexible changing groups and clients the user account cannot be used to identify a client. The Metafora platform uses a token, which is generated for the login of the first user on a client, and revoked on logout of the client, for the clear identification of the client. This token is, together with the local users, shared with all opened tools on this client with the help of URL GET parameters.

All tools can interact with each other and the platform by sending XML messages to the XMPP channels. These command messages use a universal format and can be reused. Some of the integrated tools support *referable objects* [2], which are learning artifacts, consisting from a technical point of view of an URL and some context information. The Planning Tool, PiKI and eXpresser allow the students to share the current workspace to a LASAD discussion map or the Metafora platform chat. If a workspace is shared to LASAD the discussion node is added to the open LASAD discussion map of the sharing client. This map can be identified with the help of the clients token. To create such a referable object the tools send a XML action to the command channel. The source of such a message can be any tool; microworlds can be extended with this feature without code changes in other modules. A student can

open a referable object on his or her client. This is done with the same XML command which is used by Planning Tool to start a resource card. The Metafora platform extends the URL of such messages with the clients token and users.

The loose coupling approach of flexible combination of analyses in Metafora doesn't fit to the idea of a fixed analysis dashboard. Our design is focused on integrated and embedded support [8] of the learners to allow learning support directly connected to the tools and their contexts. The integrated tool-specific and cross-tool analysis components listen to messages in logger and analysis channel and post their result to the analysis channel. This analysis results are used by different analysis views. The workbench provides a list of current landmarks of all groups in the students challenge as Breaking News. And the Reflection Tool, which is embedded in Planning Tool, shows the landmarks of the students group to support reflection on the groups approach. The integrated tools can also use feedback messages to support the students. Feedback messages are command XML actions which address some students and are shown to them by the Metafora platform or the embedded mircoworlds. Metafora supports three awareness levels for such messages (see also D5.2):

- high interruptive messages are shown as modal popup dialogs
- low interruptive messages are shown for a few seconds into the lower right corner, like Skype messages or the Growl notification system
- no interruptive messages are only added to the Notifications list

The feedback message XML command also uses a universal format and can be sent by any integrated tool. This is used from the Messaging Tool to allow users (or intelligent anaylsis agents) to generate feedback messages.

All Metafora core components have some common design ideas. They all use the XMPP Bridge to connect to the XMPP channels. The XMPP Bridge is our high level XMPP abstraction library, based on the Smack library, which can be used to send and receive XML actions quite easy. This library is also used by some of the integrated tools.

To avoid redundancy and to simplify the configuration and installation, all core components use the *metaforainit* MySQL database to get their account credentials

and server addresses. The class MysqlInitConnector, which loads the information, tries to connect to a local MySQL server. If it fails, the MySQL server on metafora.ku-eichstaett.de is used as default.

The Startup servlet, which is triggered on module load, uses this class to initialize itself and provide the information if required. The model of the *metaforainit* database is shown in Image 3. The table *server* defines the used server. Every record consists of a name, which identifies the server and its URL. If a server requires login credentials, like the MySQL server or the XMPP server, the table *account* contains this information. The account records depend on the used module. To assign module dependent values to the right module the table *modul* implies one record for each Metafora module, with the name of the module and an ID which is also used in other tables. The table *xmppchannels* saves the channels which should be joined with an account. For general data like module names for XML commands, the table *general* is used as key-value store.

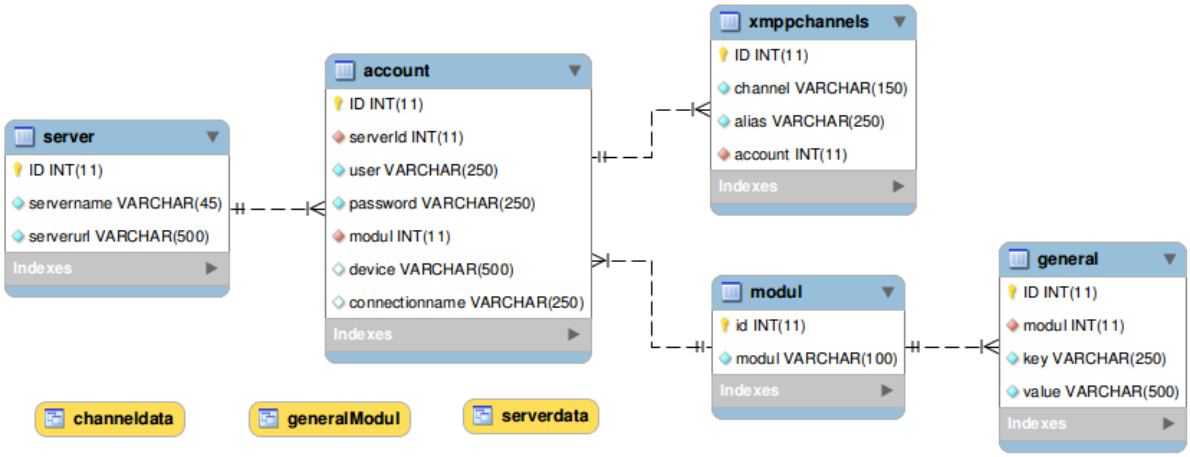

Image 3: The database "metaforainit": central storage for URLs and account credentials.

2.3. Security

The security concept for Metafora, as a web-based open collaborative learning environment, takes different aspects into account. On the one hand we have to protect the platform as well as the stored data against attacks from the outside, on the other hand we have to share most of the data with the integrated tools. Another aspect of the Metafora security concept is the users. We want to allow all users to collaborate with each other, but we also want to protect the work of the users against

vandalism. Our security concept is based on the Metafora related german bachelor thesis *Entwicklung eines Konzepts und Realisierung für die Verschlüsselung von Daten innerhalb einer webbasierten kollaborativen Lernumgebung*[9] [3] supervised by the KUEI team:

A weak spot of distributed web-based platforms are the communication channels of the platform. A well-known example for an attack on the communication channel between the client and the platform is firesheep[10], a firefox plugin which allows the user to hijack[11] Facebook sessions of other users. This was done by reading the Facebook session id from the unencrypted network traffic. In Metafora we use HTTPS for the communication with the client[12]. The HTTPS protocol is an extension of HTTP which encrypts the communication to and from the server with SSL[13]. The microworlds in Metafora are embedded with the help of inline frame HTML elements. This inline frames are initialized with an URL and the browser opens a connection to this URL and renders the content. If an embedded tool offers a HTTPS connection it is used by Metafora to protect the user's communication with the tool.

Less critical communication channels are the XMPP connections and the database connections, which were introduced in D4.1. A default Metafora installation runs the database servers on the same machine as the tomcat server, so no database communication is sent over the network, because only the server side Metafora core modules communicate with the databases. The default installation of Metafora uses no encryption for the communication with the MySQL database, because Metafora needs a lot of write and read access on this database. To enable encryption for this communication would cost a lot of CPU load. If the MySQL database is not hosted on the same machine, the encryption for the database connection can easily by enabled. Every database communication of Metafora core modules is encapsulated, for each module, in a class *MysqlConnector*. This class uses the official MySQL

[9] Development and realisation of a concept for the encryption of data in a web-based collaborative environment

[10] http://en.wikipedia.org/wiki/Firesheep

[11] http://en.wikipedia.org/wiki/Session_hijacking

[12] https://en.wikipedia.org/wiki/HTTP_Secure

[13] http://de.wikipedia.org/wiki/Transport_Layer_Security

Connector/J[14] which supports SSL encryption for the connection and the encryption can be enabled by changing a flag in this class. Our document store CouchDB uses a HTTP interface for communication with the database. The newer versions of the CouchDB support also HTTPS for this interface and also the library we use, couchdb4j[15], supports SSL. A default Metafora installation uses HTTPS for this connection.

Metafora uses XMPP multi-user chats for its blackboard architecture. The *eJabbered*[16] XMPP server - which is also a part of the default installation - supports SSL encryption for the client communication. Tools which use the *XMPP Bridge* have this encrypted communication by default, thus all Metafora core components and most of the integrated microworlds use encrypted communication. Users should be aware that everyone with a valid XMPP user account for this server can join the Metafora multi-user chats, read all current messages and post messages. This is a trade off to our open design and fits to our open platform approach.

For a web-based platform like Metafora the whole server setup and not only the platform components are important for the security of the platform. A default Metafora installation contains *shorewall*[17] as firewall. This firewall is based on the iptables[18] packet filter, which is part of every Linux system. For a default Metafora installation only the *Apache* server, the *eJabbered* XMPP server and the *ssh* server for administration, is reachable. The *Tomcat* server is connected to the *Apache* server and doesn't need to be directly reachable. This protects the server from attacks against any other components, like the database servers.

The most private user information is the user password. Even if this password should be a unique password for Metafora most users reuse passwords of other services. Metafora stores information about this password and logs in the user to integrated

[14] http://dev.mysql.com/downloads/connector/j/

[15] https://code.google.com/p/couchdb4j/

[16] http://en.wikipedia.org/wiki/Ejabberd

[17] http://en.wikipedia.org/wiki/Shorewall

[18] http://en.wikipedia.org/wiki/Iptables

tools. To protect the password, Metafora calculates a *MD5*[19] and a salted *SHA256*[20] hash of it on the client side and uses those hash values, but it never sends or stores the plain text password. The salted SHA256 hash is stored into the Metafora database because MD5 is vulnerable for rainbow table[21] attacks. The MD5 hash however is used for the login in some of the tools, since it wasn't possible to use SHA256 for all tools, due to compatibility reasons, major code changes and loss of user data which existed before the Metafora project.

Metafora, as a collaborative platform, needs both protection from external attacks and vandalism by registered users: Because of the lack of rights management and access restrictions, which allows all users to collaborate easily, users can damage or destroy the work of others. Also features like the file upload can be used to share illegal content. Metafora has no integrated protection against vandalism, but all information is stored in the databases. The default Metafora setup contains backup scripts, which do incremental backups of the MySQL databases, containing the plans and backups of the CouchDB document store. Those backup scripts are triggered every night automatically by a cronjob. For the CouchDB, incremental backup isn't necessary, since this database automatically stores versions on document uploads and users cannot delete uploaded documents. With the help of these versions and the incremental backup of the MySQL database, old versions of plans and uploaded documents can be restored from the system administrator with little effort. The system administrator can also remove documents from the document store, if they were uploaded accidentally or contain illegal content.

For the Metafora demo instance[22], we use a script which restores the platform every two weeks to the initial demo state. This time range is long enough for small experiments with a few school lessons but avoids that the platform gets spammed with content. If someone requires more time we can also pause the automatic reset on request.

[19] http://en.wikipedia.org/wiki/Md5

[20] http://en.wikipedia.org/wiki/Sha256

[21] http://en.wikipedia.org/wiki/Rainbow_tables

[22] https://metafora-project.info

2.4. Flexibility and openness

In the three years' duration of the Metafora project, we developed a technical system which is designed as an independent and heterogeneous framework. The framework defines interfaces to integrate applications with different underlying technologies, enabling seamless transition between the tools, thus it evokes the feeling of a desktop like application. Developed as a web environment, the Metafora platform can be used independent of the locality via a web browser.

One major aspect of the flexibility and openness of the system is the availability in different languages. As described in D4.1 the Metafora platform implements an internationalization mechanism to support the platform in several languages. The supported languages of Metafora are:

- English
- Greek
- Spain
- Hebrew
- Portuguese
- German
- Chinese

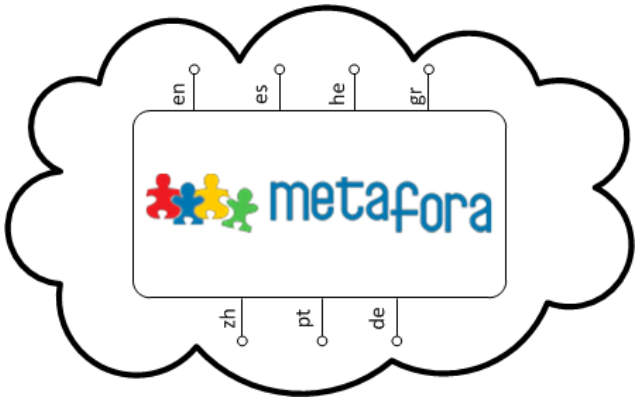

Image 4: Supported languages of the Metafora platform, addressable via URL parameter.

Through the GWT internationalization constants all strings shown in the graphical user interface of the platform were translated in the supported languages. The translated constant strings are sourced out into a MySQL database for persistent storage. When compiling a version of the system, the strings are loaded from this database and inserted into the application. Afterwards the application can be called in any of the associated languages via a parameter in the URL. Through sourcing out the strings of the graphical user interface, it is very easy to change the translation on the fly without major technical effort or knowledge. New entries for a language can be added as well as complete additional languages. Through this mechanism we were able to support the system internationalized in different languages for multi-national experimentation. The following image shows the Metafora platform in Chinese.

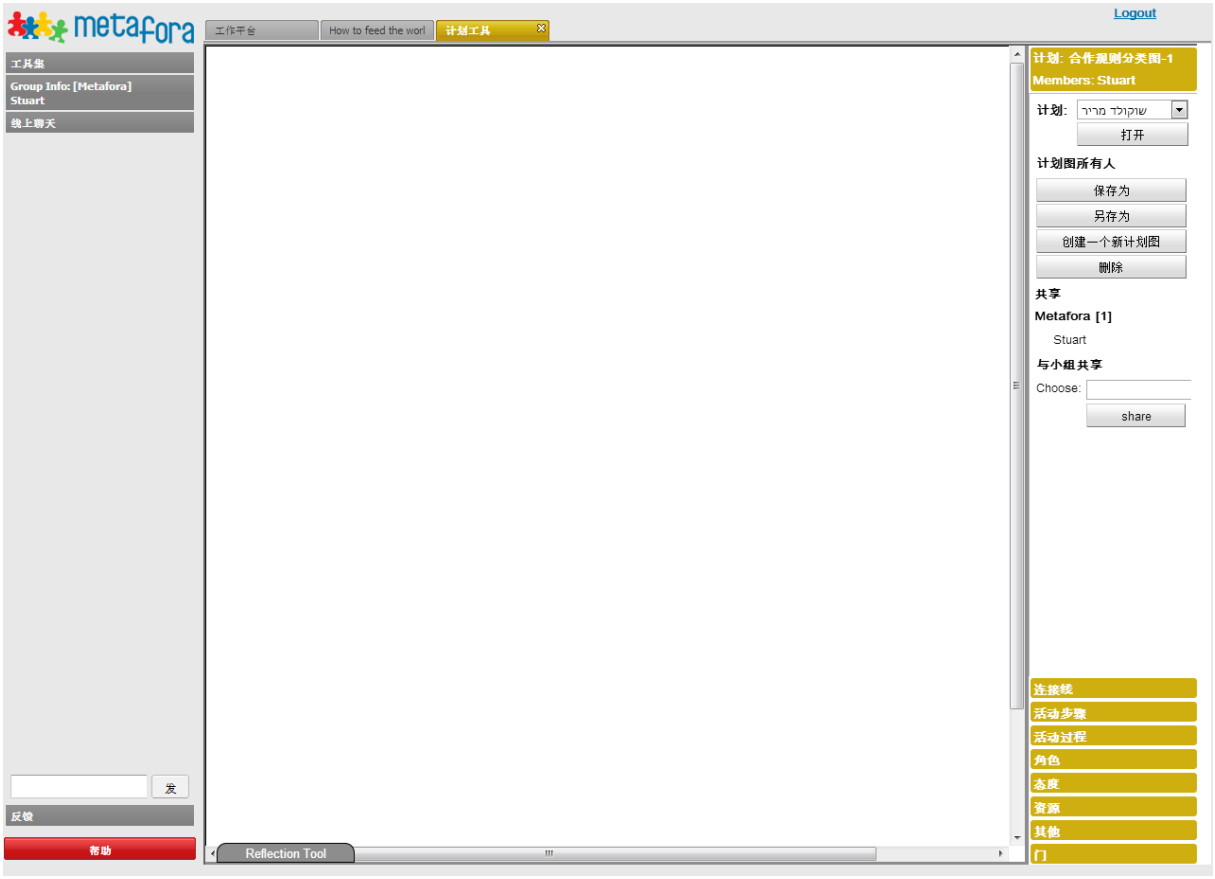

Image 5: Metafora platform in Chinese.

Another aspect of the flexibility and openness of the system is the unified semantic integration of the tools. Although all tools depend upon different technologies, they were all integrated within the same consistent UI design (in close cooperation with WP6), as for example in the visual language of the Planning Tool. The visual language in the Planning Tool offers a category for resources. In this category, every tool of Metafora is represented by its own element as a resource card. Visually the resource card is designed as any other visual language element, but with the feature to serve as entrance to the tool. Learners can start enacting right from the plan by selecting the “start using” functionality on the resource card, which opens the associated tool, independent of its underlying technology, as well as server-based or client-based.

Enacting through resource cards from the Planning Tool enables the storage and retrieval of compiled artifacts to a plan. Compiled artifacts are for example discussion

maps, Planning Tool plans or specific work states in a microworld which were previously compiled by the students. Learners can save their work from the tool to the resource card and thereby make it accessible to other learners.

Since Metafora embeds web-based but also non-web-based tools, two mechanisms were developed to enable the storage and retrieval of artifacts to a resource card. Artifacts of web-based applications with their own persistent data storage can be accessed by a reference in the tool. A mediating agent, as already explained in section 2.2, between the Planning Tool and the resource tool requests the artifact as for example a discussion map and allocates the reference parameter to the Planning Tool, which updates the resource card with this reference. Afterwards the artifact will be loaded when enacting through the resource card by passing the reference parameter to the tool.

In contrast, non-web-based tools without their own persistent storage use a service, offered by the Metafora system, which was also already mentioned in section 2.2, to save data persistently and homogeneously into a database. If a non-web-based tool as for example PiKI saves the current state of the compiled work, it generates a document with the necessary information about the work. Afterwards PiKI uploads the document into the CouchDB of Metafora via the Metafora Upload Service. The upload service returns a reference ID to the document in the database to PiKI, which then passes forward the reference to the Planning Tool. Subsequently the Planning Tool updates the resource card of PiKI with this reference. Whenever using the tool through the resource card, the reference will be passed to PiKI and the compiled work be loaded. Thus the saved state is accessible through the resource card in the Planning Tool plan.

Image 6: Enacting a saved PiKI artifact through the resource card.

Through the flexible architecture, presented in section 2.2, of Metafora, different tools with a wide variety of technologies were integrated within the same uniform UI design. It offers a persistent and homogeneous data storage for non-web-based tools and enables the easy integration of further tools, independent of their underlying technology. With the resource cards in the Planning Tool, new tools can be embedded into the visual language of Metafora and thus be used to plan learning activities in a learning scenario. Chapter 3.4 will present *CAVilLag*, an application with which the visual language of the Planning Tool can be flexibly designed according to fit arbitrary needs of a learning scenario.

3. Tools

In this section we will introduce the tools of the final Metafora system. First we will present the embedding framework of the platform, which is called *Home*. Afterwards the *Planning Tool* will be shown which integrates the visual language of Metafora. Following that, the *Reflection Tool*, a tool to support the reflection of learners during their whole learning process and *CAVilLag*, a tool to design the visual language according to the needs of a learning scenario will be presented. The chapter ends with the description of the *Workbench* of Metafora.

3.1. Home

The Home is the central integrating component of the Metafora system, a web-application developed with GWT[23]. It serves as a visual container for the Microworlds and as an entry point with login management for users to the features, Metafora provides.

On server-start the home initializes XMPP-connections to the logger, the analysis and the command-channel and loads all tool-URLs that are later on needed to launch the Tools on client side with the MysqlInitConnector and keeps them cached on server-side. Afterwards it starts the UserManager that manages all user- and group-information needed for user logins.

3.1.1. Login

When a user starts the Home-client, he is led to a login screen to enter his username and password. If cookies already contain a token and session information, the old session is restored and the login-process omitted. Otherwise a new token is generated, using the current time and a random number, and registered at the server. The user then selects group, challenge and Planning Tool map from lists with recent selections highlighted, each submission getting sent to the Home-server via GWT-AsyncCallbacks. After the login process, the Home's graphical user interface is displayed as in the illustration below.

[23] http://www.gwtproject.org/

3.1.2. Graphical User Interface

The Contentpanel with AccordionLayout[24] on the left contains the Tools section, which provides links to the Metafora Tools, that will be used in Metafora, as well as the Group Info, Chat, Notification Area and the Help-button.

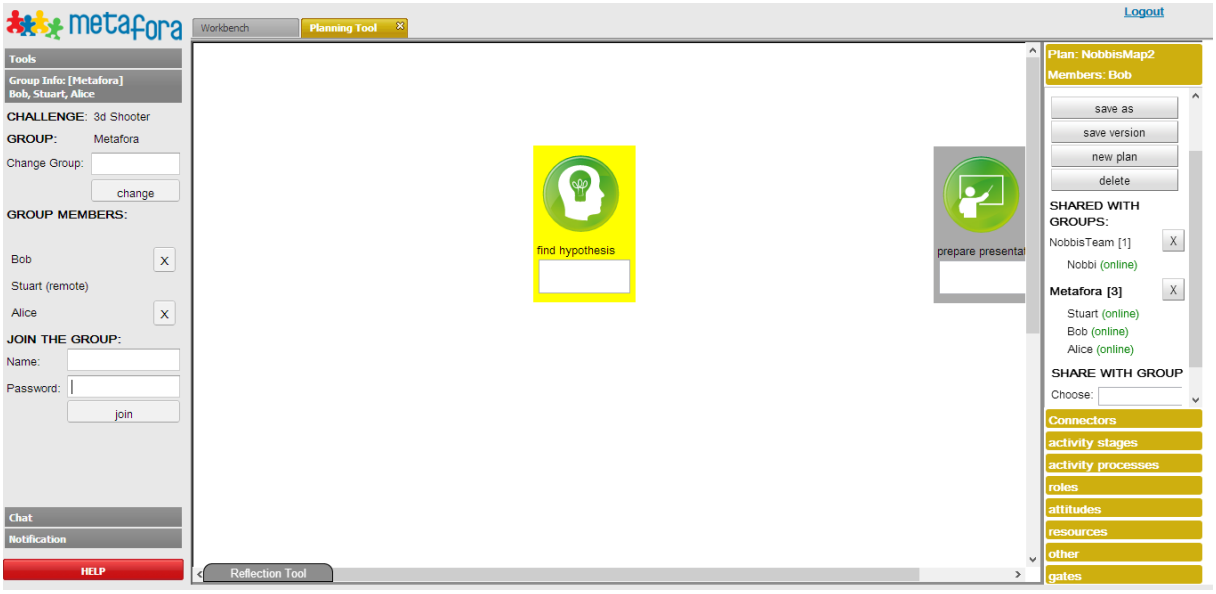

Image 7: Screenshot of the Home

The right part of the UI is occupied by the logout-button and a TabPanel that contains all Tools, that have been opened as tabs and represents the main working area for learners in Metafora. By launching the respective creation messages into the command channel, Tools can be opened remotely for specific users[25] as well as by the user directly in a transient state via clicking on a link in the Tools-section.

As already covered in chapter 2.2 Group Info provides social awareness and group manipulation features, giving the user an overview of who in his group is online at any time while he is online. Chat and Notification area help students communicate and raise attention, each area displaying one or two red arrows to generate awareness, when messages have come in but have not yet been read. The Help button provides a contextualized dialog in which the user looking for help can specify and issue a help-request to other learners.

3.1.3. Dependencies

The Home uses several libraries that are needed for it to run properly. These are in

[24] http://www.sencha.com/products/gxt/, http://dev.sencha.com/deploy/gxt-2.2.4/docs/api/

[25] see chapter 2.2

particular:

- gwt-crypto[26]: used for password-encryption during the login process
- gwteventservice[27]: Server push-library, that is used, to send chat-messages, notifications and open-Tool-commands to all registered clients
- gxt: Contains several widgets, that are used to build the graphical user interface.
- log4j[28]: Used for logging
- mysqlconnector: Needed for the connection to the MySQL database during server initialization to get all user- and group-information
- XMPP Bridge: Manages the connections to the three XMPP-channels

3.2. Planning Tool

The Planning Tool is the central L2L2 component of the Metafora platform. It is a collaborative, graphical editor which allows the students to plan their activities in a learning scenario with the help of cards of different categories, like *activity stages* or *resources*, and different edge types with meanings like *is next to* or *is needed for*. The Planning Tool also starts the required mircoworlds and visualizes the progress of the collaboratively created plan. Image 8 shows a screenshot of the Planning Tool.

As most of the GWT Metafora components, the Planning Tool has a class StartupServlet which loads available settings like server URLs and account names from the *metaforainit* database. This is done only once when the module is started and the information are cached form StartupServlet. For details about this configuration mechanism see section 2.2.

[26] https://code.google.com/p/gwt-crypto/

[27] https://code.google.com/p/gwteventservice/

[28] http://logging.apache.org/log4j/2.x/

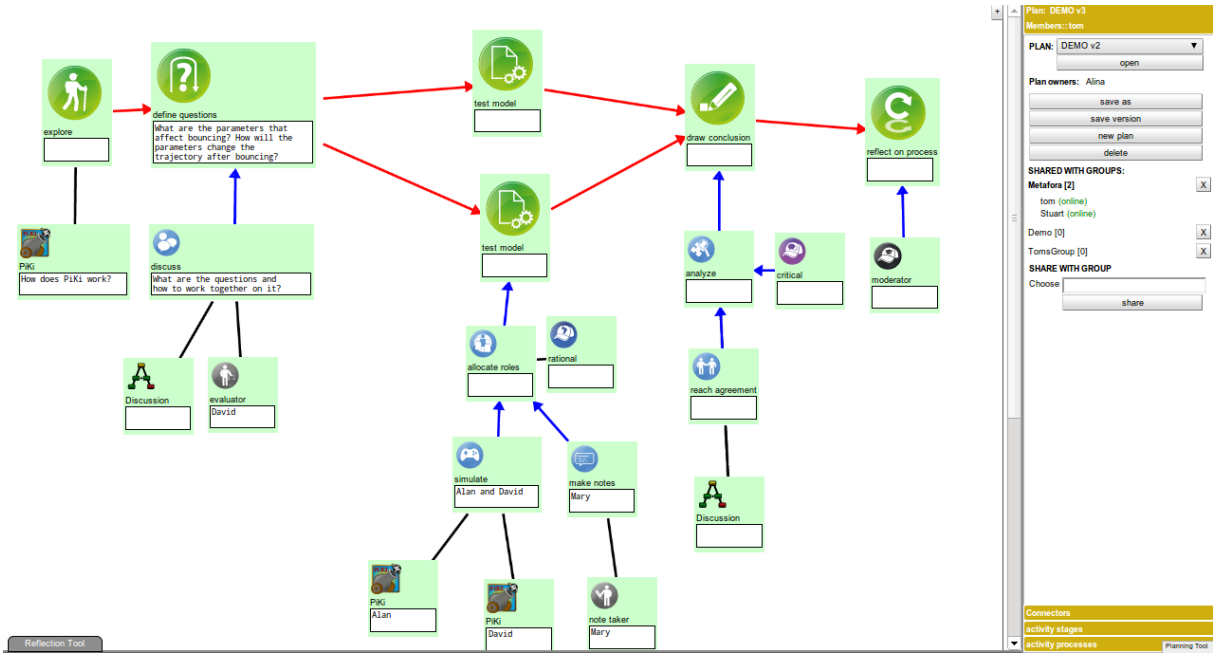

Image 8: Screenshot of the Planning Tool.

The Planning Tool interface is split into two sections. A small sidebar on the right side provides plan information and available cards, sorted by categories, which can be dragged to a large planning space. The available cards and the order of the cards depend on the challenge and can be configured with CAVilLAG (cf. section 3.4). The database *metafora* contains all available cards and their different localizations. Image 9 shows the database model for the cards. When a client loads Planning Tool an asynchronous JavaScript server call is started which requests the available cards for the selected language and challenge. This updates the last used tag of the challenge and triggers the NodeTypeManager to load the cards. The NodeTypeManager loads all available cards for the selected language and the challenge template from the challenge table, which is also part of the database *metafora*. The localization of the cards is done with the help of the SQL query and the filtering and order of cards is done from NodeTypeManager. Metafora uses a MySQL database and Planning Tool encapsulates all database requests in the class MysqlConnector, which is based on the library *MySQL Connector/J*, version 5.1.14. This encapsulation allows replacing the MySQL database with another database quite easily. The database request uses the table *iconrelative* if such a table is available. This table should contain relative URLs for the icons. If the icon URL is no absolute URL the NodeTypManager completes the URL with the Apache server of StartupServlet. If the card is connected

to a microworld, this microworld can be linked with the help of the *toolurl* column. The NodeTypeManager implements a mechanism which allows to complete this toolurl with an *metaforainit* value. If the URL starts with a pattern like *server* this part is replaced with the URL of the server of *metaforainit*.

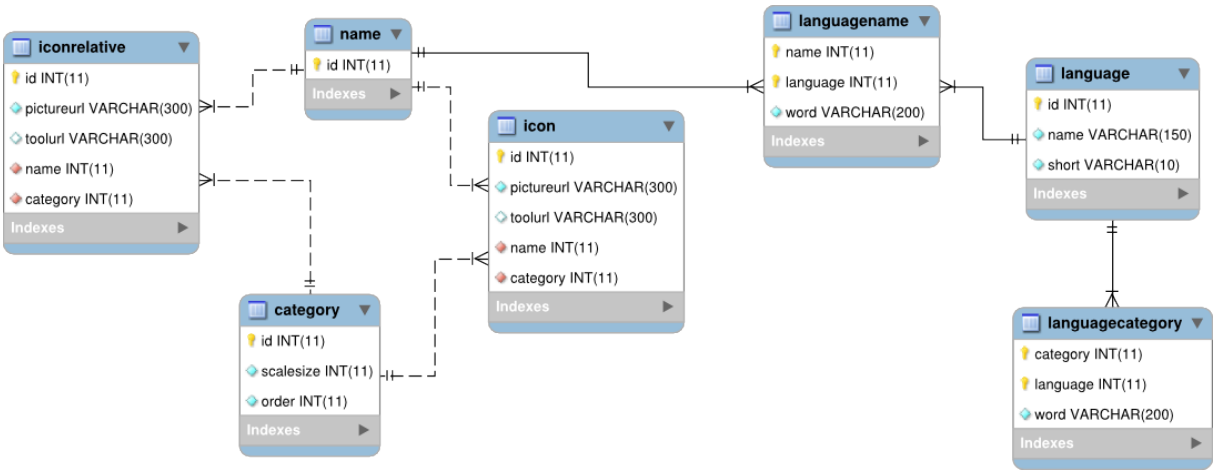

Image 9: The model for the Planning Tool cards in the database „metafora".

Parameters like the selected language and initial Planning Tool plan can be set with the help of the GET parameters[29]. Required parameters are the *token*[30], the *user* and the *challengeId*. The *challengeId* is a unique identification number for the challenge. The user is, like all GET parameters, URL encoded to support international alphabets (e.g. Hebrew, Greek); this means all no-ascii-characters are percent-encoded[31]. The optional parameters are *ptMap*, *otherUser*, *centerNode*, *challengeName* and *cavillag*. The parameter *ptMap* should contain a Planning Tool plan name. If no plan is given, the plan *default* will be opened. If this parameter contains a plan which doesn't exist it will be created. With the help of parameters which start with the name *otherUser*, more than one local user can be given. The parameter *challengeName* defines the name of the current challenge, which is used for the action XML messages. This parameter should contain the name of the challenge, which is selected with *challengeId*. The last parameter *cavillag*, can be true or false and the default value is false. If this parameter is true, the last used tag of the challenge will not be updated.

[29] https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[30] Please take a look at section architecture of the Metafora platform for details.

[31] http://www.ietf.org/rfc/rfc3986.txt

This tag is required to protect challenges from editing via CAVilLag as explained in section 3.4.

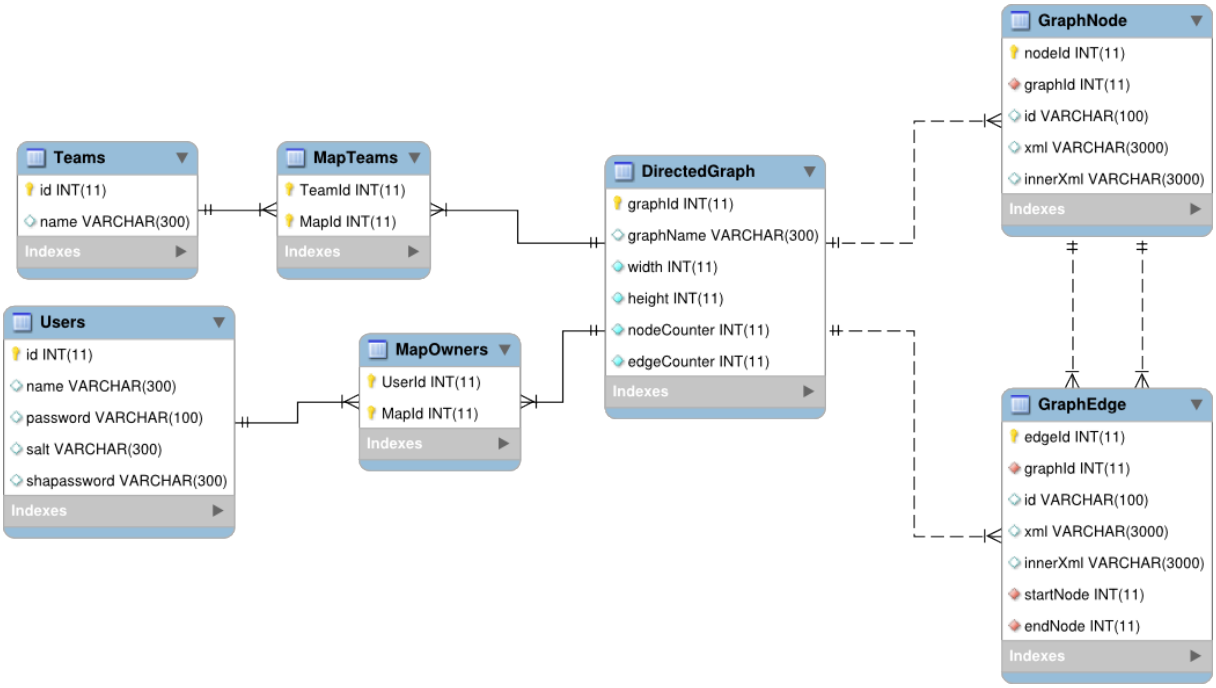

Image 10: The model for plans in the database „metafora“.

In the Planning Tool users can create, load and delete plans in the plan section of the side bar. Planning Tool uses an in-memory data model which is directly coupled with a database. This means, if a user changes some plan information, this modification is handled by the Java object which represents this information fragment and triggers a SQL call which updates the database. If the module is restarted an existing plan is reloaded from the database when a user requests this plan.

If a user creates a plan, this plan is owned by the user and shared with the current group of the user. Those relations are stored in the tables *MapTeams* and *MapOwners* of the database *metafora*. Image 10 shows the data model of the plans. The tables *Users* and *Teams* are unrelated because in Metafora every user can join any team. The list of available plans in the Planning Tool contains only plans which are shared with the current group of the user and plans which are owned by this user. If a user shares a plan with another group also users which joined the other group can find this plan in the list of available plans. Every user who can see a plan can share it with other groups but only the user who created the plan owns the plan.

Planning Tool shows a list of the groups that share the plan and the users who are working with the plan at the moment in the plan section of the side bar.

Users can create snapshots of a plan with the help of save version. This function creates an XML document which describes the current state of the plan and uploads it to the document store. The XML description of the plan is created by class GraphManager on server side and the upload is encapsulated in class DocUploadService. We use CouchDB as document store and DocUploadService, which is based on the *couchdb4j*[32] library.

Users can change a plan by adding, editing or deleting cards. A new card is added to the plan if a user drags a card from the sidebar in the plan area. This drag and drop feature is realized with the help of the *gwt-dnd*[33] library, as reported in D4.1. If a drag event removes a card form the side bar this card is cloned. It allows that a user can create multiple cards of the same type. The user can drag the card to any place in the plan. As drag handle of the icon on the card is used. The plan area can be extended by the user when there isn't enough space. To extend the area in horizontal space users can use the plus button in the upper right corner of the plan area. Also, a resize in vertical space is triggered if a card is pushed to the lower border of the plan. There are three types of card modifications. A card can be moved with the help of drag gestures, the state of the card can be changed and the description of a card can be changed. If users move cards at the same time, the last user wins. Possible states for a Planning Tool card are “started” (yellow cards) and “done” (green cards) and the state of a card can be selected with the help of the context menu of the card. If users modify the state of the card at the same time, the last modification wins. If users edit the description of a card at the same time we support social conflict handling to stimulate L2L2, which is described in more detail in [4]. And finally users can delete a card with the help of the context menu of the card.

Planning Tool supports different types of edges. The type of the next edges can be selected in the sidebar. Edges always connect to different cards, so the user has to

[32] https://code.google.com/p/couchdb4j/

[33] https://code.google.com/p/gwt-dnd/

select two cards for creation of an edge. A card is selected with a click on the icon, which also is the drag handle. If the drag is less than a few pixels, it is interpreted as a click event. A selected card is highlighted in a light red and can be deselected with a second click. If a card is already selected, the class EdgeHandler creates a new edge from the selected card to the clicked card with the selected type. In Planning Tool edges are not single graphical elements. We use a HTML5 canvas[34] as glass pane to draw all edges.

Furthermore the Planning Tool supports collaborative planning, so we have to update the plan on all clients if it is changed. To reduce network traffic and server load we use *long term polling*[35] as serverpush technology. We also evaluated newer solutions like HTML sockets, but these were barely supported with older browser releases and didn't work well with most of the school network infrastructures. The server push in the Planning Tool is based on the *GWTEventService*[36] library. With the help of this library the server push can be used like an observer pattern[37], which means clients register for domains they need and messages are added to domains. It also allows multiple client instances in the same browser session, which isn't supported by most of the other libraries and is required for Planning Tool to support multiple open plans.

To communicate with the Metafora platform, the Planning Tool uses XMPP connections to the analysis, command and logger channels. The connection to the logger channel is used for logging all user interactions like moving of a card. The connection to the analysis channel is used to provide information about high level user interactions like starting or finishing a card/learning activity. The command channel is used for sending open tool commands to the embedding Home application, when a student starts a resource card. The Planning Tool also listens for *modifyNodeUrl* commands on the command channel, which are used to update the tool URLs for existing resource cards. Integrated microworlds use this to link

[34] http://en.wikipedia.org/wiki/Canvas_element
http://www.gwtproject.org/javadoc/latest/index.html?com/google/gwt/canvas/client/Canvas.html

[35] http://en.wikipedia.org/wiki/Push_technology

[36] https://code.google.com/p/gwteventservice/

[37] https://code.google.com/p/gwteventservice/

resource cards to instances of mircoworlds, so it is possible for students to resume their microworld sessions.

3.3. Reflection Tool - An embedded view of groups' learning processes

The Reflection Tool is a visually embedded (in contrast to external analytics tools in the dashboard style) reflection component for the learning whole process of a group. It supports a group of students to reflect on their activities in the learning scenario. This implies that learning events of other groups are not part of this view (cf. to the coupled context explained in section 2.1.1) and that the events of the group of the students are reduced to important steps of their process. The Reflection Tool uses the analysis multi-user chat as data source, which contains the results of analysis tools and important events of microworlds. To keep the amount of data small enough to be useful and to keep the focus on the learning process of the students, indicators[38] and landmarks[39] of other groups and other challenges are dropped. An important requirement for the visualization of the embedded analysis is that it has to be intuitive. Our Reflection tool uses a timeline presentation for the landmarks which is a well known representation for the students from diagrams, sports tickers etc. A more detailed description and evaluation of the Reflection Tool will be presented in the Learning Analytics Workshop at DeLFI conference and can be found in [5]. Image 11 shows a screenshot of the Reflection Tool.

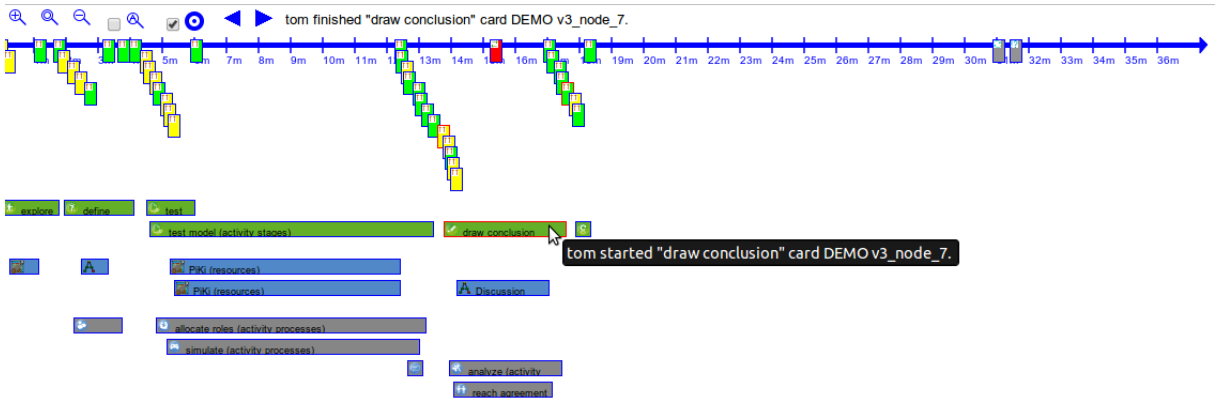

Image 11: Screenshot of the Reflection Tool.

[38] Indicators are less important steps of the students learning process, like choosing of a set of parameters for a simulation.

[39] Landmarks are important steps of the learning process, like start or finish of an activity.

The Reflection Tool is implemented as GWT application, which is useful due to the need of only a small subset of old data from the analysis channel used for its visualization: The implementation as GWT application allows calculating the subset of the channels' history for each group-and-challenge combination (coupled context) only once when it is requested for the first time, and caches the result. Another advantage of the implementation as GWT based application is that the Reflection Tool can follow the Metafora core component design and uses the *metaforainit* database to get the required server URLs and account credentials. The only two required GET parameters[40] for Reflection Tool are *challengeId* and *groupId*. The combination of those parameters defines which landmarks are shown. The parameter *challengeId* has to be a valid challenge id, which is the primary key of the table *challenge* of the database *metafora*. Details of the table *challenge*, which stores the information of all challenges, is shown in Image 12. The parameter *groupId* has to be a valid URL-encoded[41] group name.

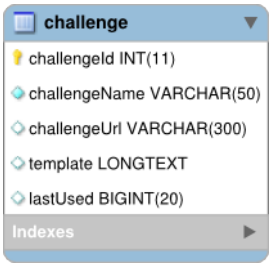

Image 12: The model for challenges in the database „metafora".

Because the Reflection Tool is not a collaborative tool and the amount of landmarks for a combination of group and challenge for a time frame of a few seconds is low, we can use polling to update the client instances with new landmarks. With the help of a GWT Timer[42] every Reflection Tool client side starts every 2.5 seconds an asynchronous request for new landmarks. Such a request is forwarded onto the server side of the application, to the class MessageEvaluator, which handles the landmark data. This class also filters and parses new XML actions[43] from the

[40] https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[41] http://www.ietf.org/rfc/rfc3986.txt

[42] http://www.gwtproject.org/javadoc/latest/com/google/gwt/user/client/Timer.html

[43] Please take a look at section architecture of the Metafora platform for details.

analysis channel and stores the landmark information as LandmarkData. The class LandmarkData is serialisable, shared with the client side and is used as container to transfer the landmarks to the clients. On the client side, this information is used to render the landmarks as cards on the timeline. To make it easy to find an event and support the understanding of the process, the landmarks are color coded. Every microworld gets its own color, help requests are colored in red and the cards contain the icon of the *L2L2* category. If a student hovers over a landmark the description of the landmark is shown as tool tip text. If the landmark is from the Planning Tool - in addition to the usual landmark information - some Planning Tool specific data is evaluated. Those are the state, the category and the name of the card and the URL of the cards' icon. To visualize the students learning process the landmarks for "start using" and "done" states of Planning Tool cards are used to create a bar (similar to a project activity bar in a Gantt chart) from the "start using" landmark to the "done" landmark in parallel to the timeline. This landmark also gets the special colors yellow for start using and green for done. The created bar is decorated with the icon, name and category of the card as visual clue. To support the students in understanding their process and its visualization the bars are linked to the landmarks of the card. If a student hovers with the mouse over a bar, the bar and all landmarks of the cards are highlighted with a red frame and the description of the first landmark is shown as tool tip. This is visible in Image 11 for a draw conclusion card from plan *DEMO v3*.

The time line in the Reflection Tool is realized with the help of a HTML 5 canvas[44]. The labeling of the timeline depends on the length of the visible time range. The minimum time length between two ticks is a second and the maximum time length between two ticks is a day. The graphical representation of the landmarks and landmark bars is implemented with the help of GWT HTML[45], that allows to register mouse over and click listeners with native DOM events[46]. Those elements use absolute positions and are layouted in the class Timeline with a method we implemented for this purpose. The layout method is triggered when the timeline grows, which is done with a timer every 5 seconds. For this layout the local system

[44] http://en.wikipedia.org/wiki/Canvas_element

[45] http://www.gwtproject.org/javadoc/latest/com/google/gwt/user/client/ui/HTML.html

[46] http://en.wikipedia.org/wiki/DOM_events

time of the client is used as current time. This can cause problems if the local time and remote time, where the message was created, differ. To avoid showing messages in future, the timeline manages an offset. If a Reflection Tool client receives a landmark which has a time stamp from the future, the offset to the local time is calculated and this offset is used to layout all messages. For reflection on the process it is important to be able to see the whole process as well as to see some specific points in detail. As default setting the Reflection Tool uses an automatic zoom to show the whole learning process in the available space. This is disabled if a student uses manual zoom-in or out and the state is shown in with a checkbox. Image 11 shows the buttons for zoom-in, fit to available screen space, zoom-out and the toggle checkbox for automatic zoom in the upper left corner. A student can select a landmark by clicking on its card representation and the default enabled *keep focus* feature of the Reflection Tool - which is shown as fifth button in Image 11 - keeps this landmark in the center of the screen while zooming. A selected landmark is also highlighted with a red border and the description of the selected landmark is shown in the toolbar. The arrow buttons allow students to step to the previous or next landmark.

If a new XML action appears in the analysis channel, the Reflection Tool evaluates the action and keeps the information as LandmarkData in memory. With this implementation all landmarks, which were produced after starting the Reflection Tool module (i.e. synchronously) are available, but no old landmarks. Yet, the historic landmarks are needed to allow reflection of the whole learning process (synchronous and asynchronous process awareness as explained in section 2.1.1). To solve this problem the Reflection Tool loads all old landmarks, if the first new landmark for a group and challenge combination appears, or when the first user requests such a combination. It is useful to load old landmarks if new ones appear to avoid long response times when the students open the Reflection Tool. To get the old landmarks, the Reflection Tool uses the history request servlet of the *Metafora Service Module*[47]. This servlet allows it to start an SQL query on the *logger* database and returns the found actions as plain text. On client side the start time of the timeline is by default the local page load time. If a client receives a landmark which is older

[47] Please take a look at section architecture of the Metafora platform for details.

than the start time of the timeline, this start time is the time of the landmark and all landmarks are layouted with respect to this new start time.

3.4. CAVilLAG

The **C**hallenge **A**uthoring tool for the **VI**suaL **LA**n**G**uage (CAVilLag) is an online-tool that allows configuring challenges with a domain specific visual language. Although Metafora's target is to enable students being in charge of deciding what model they want to create and which cards and connectors they would like to use while working with the Planning Tool, the creation of an individual visual language can make sense under certain conditions. Since there are cards available for special domain related use cases, making these cards available all the time to allow students creating individual plans can become confusing and distractive. Therefore CAVilLag has been developed as an authoring tool to allow activity designers to create a subset of the existing cards and connectors. Albeit being a stand-alone tool, CAVilLag is fully interoperable with the Metafora system and provides an online interface that can be opened in a web browser. Image 13 shows a screenshot of how cards are selected and ordered in, for the creation of a sub-set of the visual language in CAVilLag.

Image 13: Designing a sub-set of the visual language in CAVilLag.

CAVilLag provides one tab for each category of cards and one for the connectors. Users can create new challenges or change existing ones, select the cards and

connectors according to the use case in connection with the learning scenario the learners should plan for in the Planning Tool. Although it would be technically possible to change a visual language after it has been saved and used in the Planning Tool with a challenge, changing is restricted to those visual languages that have not been already used in the Planning Tool. This design decision was taken because unlimited editing would allow to delete cards even after a challenge has been opened in the Planning Tool and a card has already been used by the students in a map. Since this could lead to confusing situations and conflicts where cards that have been used before could not be used in similar context in future planning, it is strictly necessary to create a new challenge and define a new visual language instead of redefining existing ones.

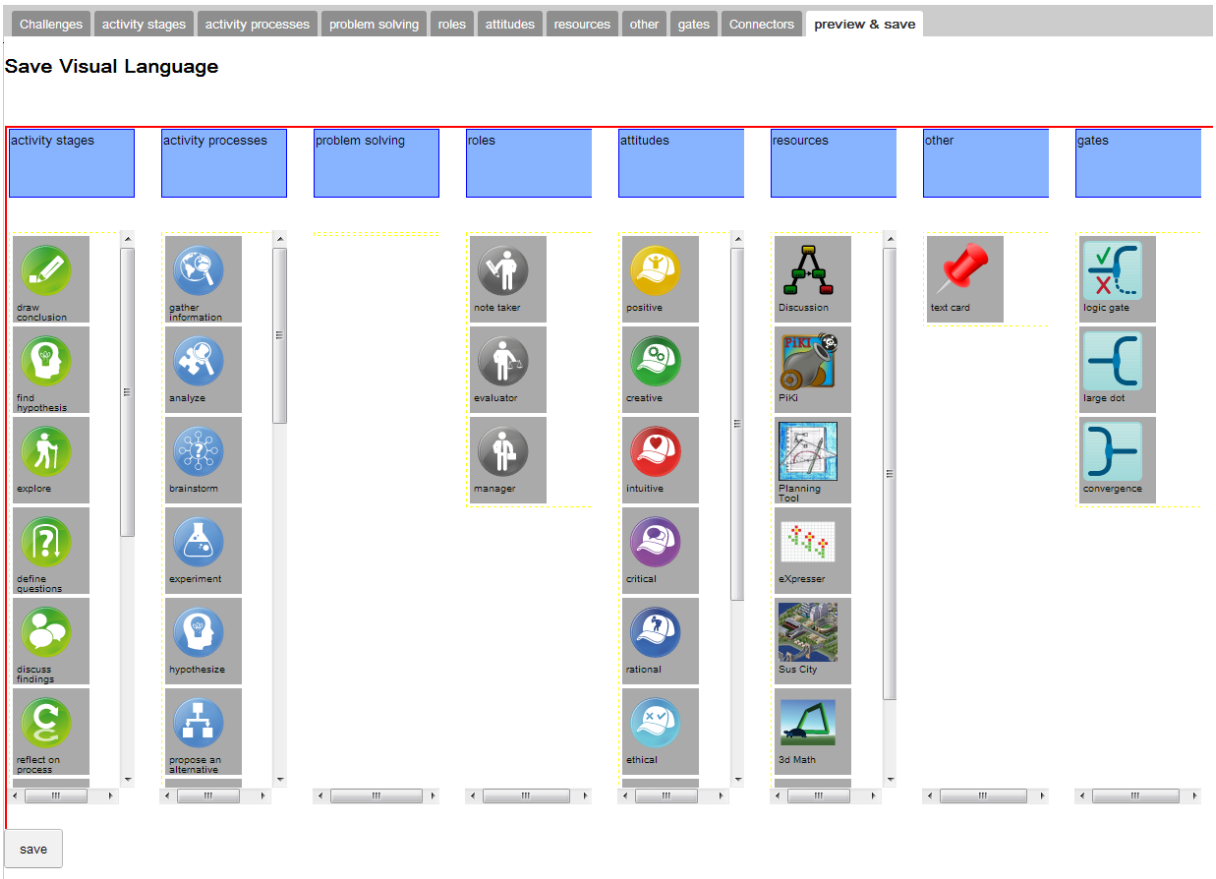

Image 14: Preview of the designed visual language in CAVilLag.

Another important aspect is the reusability of visual languages. CAVilLag supports previewing the selection of cards and connectors for each Challenge, as shown in Image 14. By using this feature the designer can decide if he or she wants to reuse a

visual language. This visual language configuration is then copied and can be fully edited before it is saved with a new challenge.

Once having saved a design of the visual language within a challenge, it is available in the Metafora platform. In the login process of the platform, users can select the challenge. Afterwards its visual language design will be loaded in the Planning Tool and thus can be used for the learning scenario. A full description of how to use CAVilLag is given in the user manual which is part of the appendix of this deliverable.

3.5. Workbench

The Workbench is a central overview component of the Metafora Platform developed with the GWT[48], meant to provide the user with information of what is currently going on in the system[49], as well as to manage documents in the CouchDB. It consists of four visible sections, shown in the graphic below:

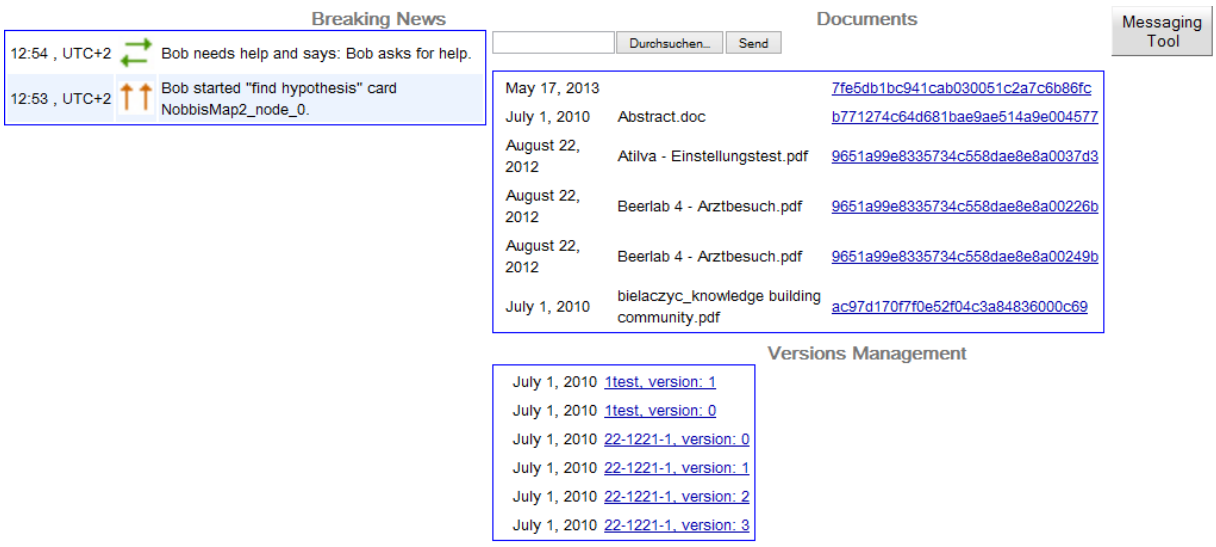

Image 15: Screenshot of the Workbench

The Breaking News-section of the Workbench displays landmarks that are interesting for the user according to chapter 2.1 and represents the awareness-part of the Workbench. Its layout was inspired by news-sections in social media such as Facebook and Twitter, to display events that happened recently and should therefore

[48] http://www.gwtproject.org/

[49] See chapter 2.1

be relevant for the user. For this we chose a table as a simple to understand visualization of landmarks. Each row stands for one landmark, the first column showing its timestamp, the second one displaying an icon representing its L2L2-Category and the third reading the content of its description tag.
The second section is the Documents-section. It contains a form for uploading documents to the CouchDB via the DocUploadServlet described later on in this chapter and a table that displays upload-time, name and CouchDB-ID of the documents that already have been uploaded.
The Versions-section is similar to the Documents, except that it shows save-date, name and version of Planning Tool maps instead of regular files.

The fourth and last component of the Workbench is the Messaging Tool button on the right. On click it opens a URL to the Home to be forwarded, enriched with the parameters necessary to display the Messaging Tool[50] (cf. D5.2) in a frame within the Workbench. Parameters passed over to the Home are locale, token, user, groupId, challengeId, challengeName and a testServer-flag.

CouchDB Connections

The Workbench document-management features use the CouchDB to store documents and therefore depend on the java-library couchdb4j[51]. On server-start two CouchDB-views are called: one that delivers names, timestamps and ids of all user-stored documents in the CouchDB and one that gets timestamps, names and versions of all Planning Tool maps. This information is enough for a simple representation of the file or version to be displayed in the Documents- and Version-section, respectively, such that the file can be retrieved afterwards. The Workbench-server keeps them cached to reduce the number of database-calls. When a Workbench-client starts, it requests all these file information via a GWT-AsyncCallback and displays it in the appropriate spaces.
A second use case involving CouchDB and the Workbench is the user uploading a file. To do so he selects a file in the Documents-section's form and submits it. The submit-method is set to post with the DocUploadServlet as target. This servlet listens

[50] the source code of the Message and Monitoring Tools are at the time of the writing available at the google code repository http://code.google.com/p/metafora-project/source/list

[51] https://code.google.com/p/couchdb4j/

to http-requests and behaves differently, depending on what method has been selected:

- Get: The servlet reads the id-parameter, opens a connection to the CouchDB, gets the respective CouchDB-document and as a response writes the data-property of the CouchDB-document, yielding the formerly uploaded file for the requester.
- Post: The servlet uses the parameters name, contentType, the data of the request and the current timestamp to create a CouchDB-document with these contents, and saves it to the CouchDB. At the same time, the Workbench-server's cache is updated to ensure consistency with the CouchDB and the new document is pushed to all clients via gwteventservice[52].

3.5.1. Dependencies

Next to gwteventservice and couchdb4j, three more libraries are worth mentioning, since they are required for the Workbench to function properly:

- log4j[53] for logging purposes.
- The XMPP Bridge[54] for the connection to the XMPP-channels. The Workbench uses the command- ,the logger-, and especially the analysis-channel, which is important for sending and receiving landmarks that can then be pushed to the clients via gwteventservice.
- commons-io[55] for processing the files in the http-requests, the Doc-UploadServlet gets.

[52] https://code.google.com/p/gwteventservice/

[53] http://logging.apache.org/log4j/2.x/

[54] see chapter 2.2

[55] http://commons.apache.org/proper/commons-io/

4. User acceptance and evaluation

In studies we have evaluated the educational effectiveness of our design principles by means of qualitative studies both in-vitro and in-vivo, i.e. in controlled lab settings and in school classrooms (reported mainly in WP3 deliverables). In the following section we will give an overview of an eye-tracking study, which was conducted to evaluate the user acceptance of the Metafora system particularly in a controlled lab setting.

4.1. MeET US Study (Metafora Eye Tracking Usability Study)

The Metafora platform provides different categories of awareness mechanisms to help the user perceiving relevant information like new chat messages, the support of their planning or to provide general information to the user in the form of *feedback messages*. To explore the perception of the different awareness mechanisms of feedback messages we have conducted a study using eye-tracking technology in a lab based situation.

There are three kinds of feedback messages (cf. also D5.2) with different level of awareness:

1. NIF = No Interruptive Feedback

   There is a visualisation that a new event occurred in form of a small arrow icon in the notification area, but the user will neither be prompted nor informed by any kind of message box or message window.

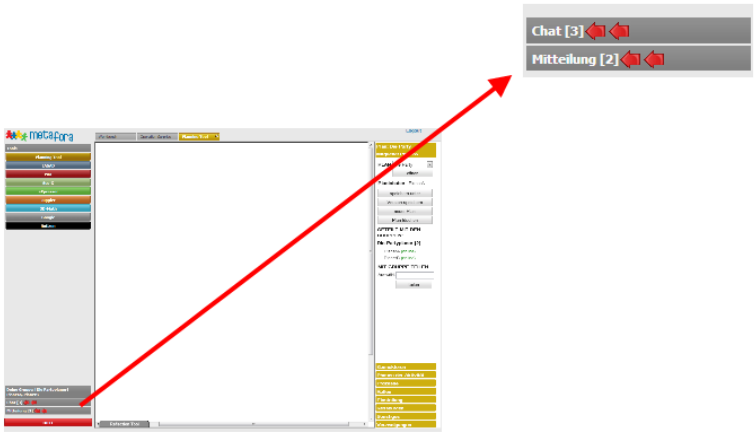

Image 16: No Interruptive Feedback message.

2. LIF = Low Interruptive Feedback

   The user will get a feedback in form of a small temporary message that will appear at the bottom of the screen, similar to the “new email” windows provided by most email clients.

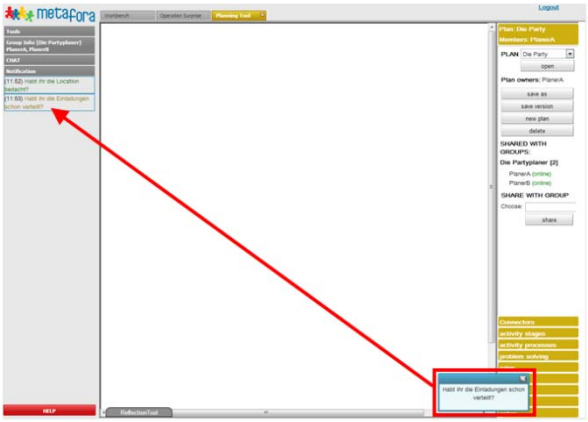

Image 17: Low Interruptive Feedback message.

3. HIF = High Interruptive Feedback

   The user will get a notification in form of a message window that will pop up in the middle of the screen and that has to be confirmed before the user can continue with the work.

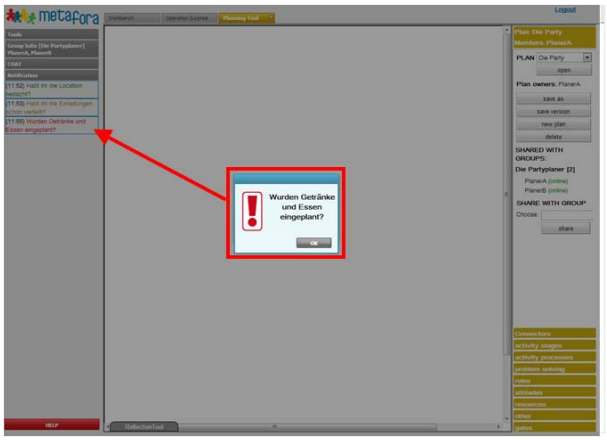

Image 18: High Interruptive Feedback message.

The study was conducted in collaboration with Prof. Andrea Kienle’s group at the University of Applied Sciences in Dortmund in their usabilty lab. In the study we have worked with dyads which were placed at two computers, located in the same room, but with a divider between the workplaces. Image 19 shows a picture of the technical

setting. The participants of the study were instructed to pretend that they are working in different locations but using a shared workspace (the Planning Tool of Metafora) where they have the possibility to see each other's work and communicate through the Metafora chat. The task the participants were asked to perform was planning a surprise birthday party for a friend. The participants got 30 minutes for the task of which the first 10 minutes were an individual planning time during which the shared workspace was used but without any communication. After 10 minutes, a High Interruptive Feedback message was launched to let the participants know that they now have permission to use the chat.

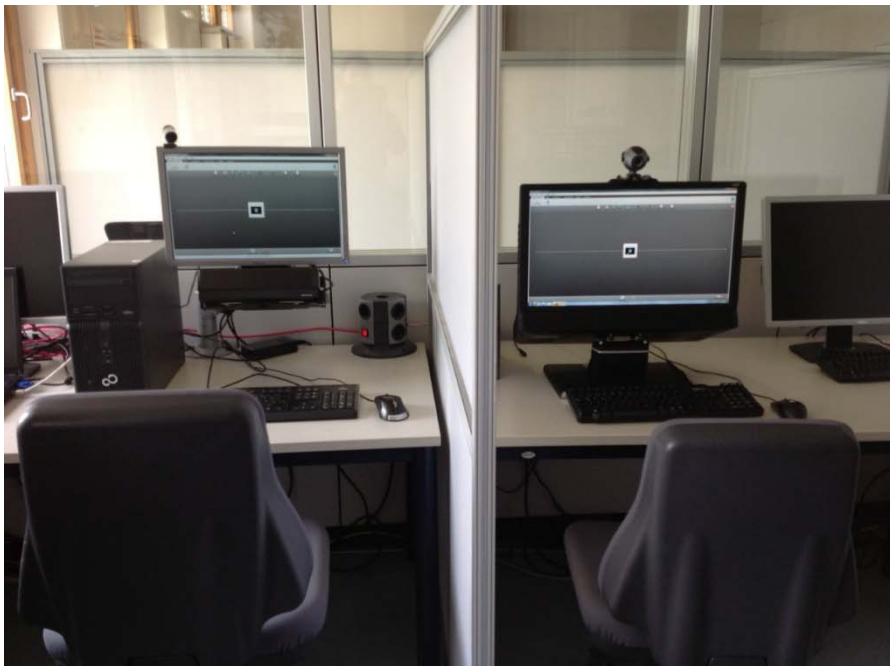

Image 19: Set up of the MeET US study.

For the execution of the experiment in a controlled way we have developed a time based script to simulate system feedback at specific timepoints and make sure that all participants receive the same message at the same time. Image 20 shows the time scale for the feedback message and what type of message was delivered (to avoid training effects we applied two different sequences of interruption levels for NIF and LIF). The high interruptive feedback at minute 10 has always been the message that the participants now have permission to use the chat, while the last HIF at minute 30 has always been the message, that the experiment is over and participants can stop.

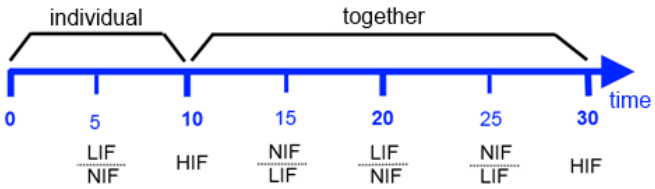

Image 20: Time based feedback script.

Through the eye-tracking technology we were able to explore the perception of the different awareness mechanisms in the Metafora platform by evaluating the eye-positions at the moment awareness evoked. Image 21 shows a screenshot of two collaborating participants during a dyad, while their eye-positions were tracked.

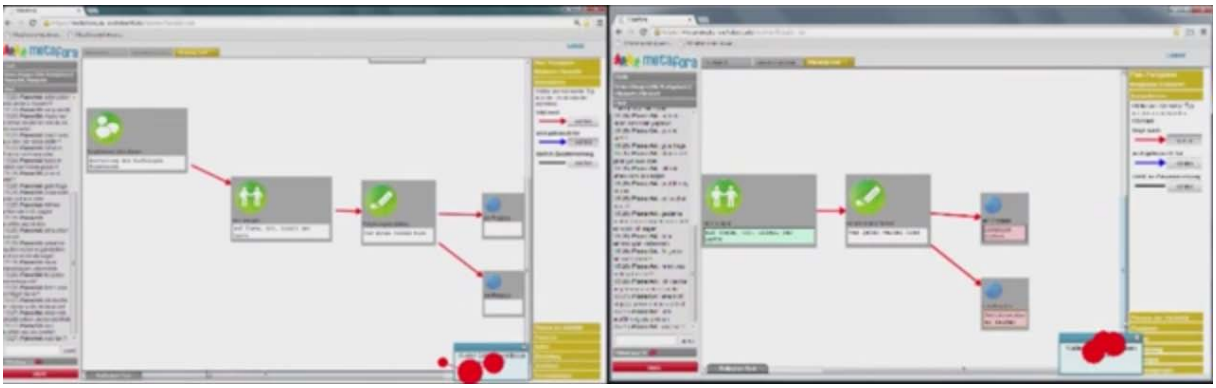

Image 21: Tracked eye positions in the MeET US study.[56]

We are currently evaluating the results of the study between the control group without awareness features and the experiment group(s) with awareness support.

As preliminary results we can state that

- High interruptive feedback was perceived immediately in its current implementation regardless of the condition
- Low interruptive feedback was perceived relatively well in both conditions with a quicker perception in the experimental group; if this was caused by a training effect of being more alert with the awareness hints has to be explored more thoroughly
- No interruptive feedback was not perceived at all in the control condition without awareness support, while in the experiment conditions it was perceived via the awareness support (arrow symbols) and inspection of the notification area by the users

[56] http://www.youtube.com/watch?v=FhtJ2kIG_Wk

- The notification area was inspected in the experiment condition pro-actively the users; some of them inspected ot repeatedly, some of them only when awareness hints had been given. We assume the the proximity of chat area and notification area made the perception of awareness hints more likely when starting to use the chat more frequently in the experiment (i.e. starting at minute 10)

More detailed analysis results on this usability study will be submitted as a conference paper and will be published also on the Metafora website.

5. Proliferation

This chapter will explicate the proliferation approach we developed to further usage and uptake of the technical Metafora system. It will explain the release of the open source code, report about the open access to the framework by having a persistently hosted system and finalize the proliferation by showing how to integrate external tools into the existing framework system.

5.1. Open source

The most important decisions for the success of an open source project are what license model and what distribution platform should be used. For the Metafora core components and also for the microworlds, a lot of libraries were used to reduce coding work. Image 22 shows an overview of the used libraries of the core components and the relevant licenses. This figure doesn't mention GWT which is licensed under a *creative commons*[57] license that only demands attribution of GWT.

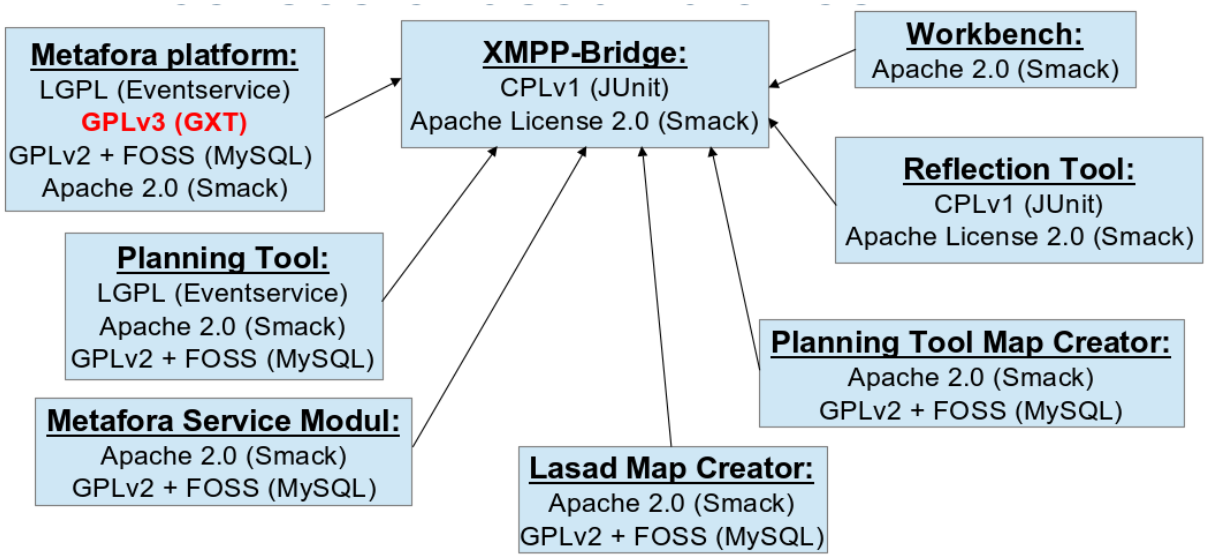

Image 22: Overview of used licensed libraries in Metafora core modules.

[57] http://creativecommons.org/licenses/by/3.0/

A well-known open source license family is the *GNU Public License*[58]. The most used versions of this library are *GPLv2* and *GPLv3*. Both libraries include a *copyleft*[59] clause, which must be seen very critical. This clause forces developers, who want to use the software, to also use the license and publish their work as open source. This causes a lot of problems, if a company wants to develop a commercial product on basis of such open source software. The GPLv2 has stronger restrictions, which make it incompatible with most of the other open source licenses; even GPLv3 and the wide spread *Apache License 2.0*[60]. The MySQL connectors, which are required for the usage of MySQL databases in software, are distributed as GPLv2 and a commercial license. Oracle, which owns MySQL, added an extension to the used GPLv2 license which is known as *Oracle FOSS exception*[61]. This extension allows to combine the used GPLv2 license with some of the other open source licenses. To increase acceptance in the open source community the GPLv3 is less restrictive and can be combined with the Apache License 2.0. This means if a project is published under a GPLv3 library, it has to use GPLv3 but it can also use libraries which are licensed with Apache License 2.0. The Apache License 2.0 contains no copyleft clause, which means code based on libraries which are licensed with Apache License 2.0 can use other licenses. The *Common Public License (CPL)*[62] has some similarities with GPL and also contains a copyleft clause, but this clause doesn't affect software which only use CPL licensed libraries. A well-known license for the CPL family is the *Eclipse Public License*.

Our aim is to initiate an open source community for further use and development of Metafora. To support the formation of an open source community we use the weakest licenses that are possible for each of the core components. Image 22 shows that the Metafora platform uses the GXT library, a user interface framework that extends GWT. For the other Metafora modules we can choose any license which is contained in the Oracle FOSS exception. We decided to use the *Berkeley Software*

[58] https://gnu.org/licenses/gpl-3.0.en.html

[59] http://en.wikipedia.org/wiki/Copyleft

[60] http://www.apache.org/licenses/LICENSE-2.0.html

[61] http://www.mysql.com/about/legal/licensing/foss-exception/

[62] http://en.wikipedia.org/wiki/Common_Public_License

Distribution license (BSD)[63], because this license is very popular with open source developers. It consists only of two sentences which can be understood by anyone. The two central statements of this license are: *"you can do with the software what you want and you do not have any warranties for the software."* Thus by using this license we allow that any company can use, improve and distribute Metafora.

The licenses of the integrated microworlds and tools are also interesting for the success of Metafora as an open source project. Those licenses affect the possible packaging and distribution of Metafora. Our AI component PlaTO [7] uses the commercial library *Jess*[64], which is a rule engine. This library is only available with a commercial license or for academic usage. This means for Metafora we can only publish the source of those tools but we cannot include the library in a Metafora package. The integrated graphical discussion tool LASAD also uses Jess and GXT, which means it has to be licensed with GPLv3 and no full featured packages are possible. The Monitoring and Messaging Tool also uses GXT components, which implies, that it has to be GPLv3 licensed.
The source code of the microworld eXpresser has 'GNU GPL v3' as its license. Besides GWT (and Java) it uses various open source libraries. It does not rely upon any proprietary software. The source is available from Google Code. (https://code.google.com/p/migen/source/checkout). eXpresser is currently hosted on the Google App Engine. The London Knowledge Lab plans to continue to pay for hosting this for at least 1 year, through a ESRC-funded knowledge transfer funding that looks into further disseminating eXpresser into UK classrooms. After that, in the absence of further funding, it will still remain available but subject to Google's 'free quota'. (https://developers.google.com/appengine/docs/quotas) This should be enough to support ten to twenty full user sessions per day.
The microworlds Juggler, 3D Math and PiKI use the *Unity*[65] framework. This framework is only available with a commercial license, but there is a free to use version. This means we can publish the source code of those microworlds, but we

[63] http://en.wikipedia.org/wiki/BSD_licenses

[64] http://www.jessrules.com/

[65] http://unity3d.com/

cannot provide packaged versions. PiKI also uses the XMPP library *agsxmpp*[66], which has a commercial license and a GPLv2 license, and the *UniSWF*[67] library, which only has a commercial license. The microworlds Juggler and 3D Math - except of Unity - use only libraries which are licensed with *GNU Lesser General Public License (LGPL)*[68], *Apache License 2.0* and *MIT*[69] license. The microworld SusCity, which is desktop based, uses also some commercial libraries, so it cannot be included in a Metafora package.

On our choice of a source code distribution platform, we have to mention two aspects. We want to ensure that Metafora is available as long as possible and we want to reach potential developers. For Metafora we decided to use two different distribution platforms. We have published the source, using the *Prose*[70] open source hosting platform and we also use the well-known and established source code platform *GitHub*[71]. Prose is a European project and develops an open source platform[72]. This platform is available for at least the next few years and is a good place to distribute open source code within European funded research projects, but it currently has very few users. To reach a broader developer community of potential users and developers, we decided to also use GitHub, which has a growing popularity. Both platforms use the distributed code versioning system *GIT*[73], which is also used for versioning of the Linux kernel. GIT works distributed, which means developers have always a local repository, and can push their work to other repositories. This allows us to use both platforms without much effort. It has also the advantage that other users can easily fork projects and send pull requests. This allows that users can develop Metafora without being a registered developer of the project.

[66] http://www.ag-software.net/agsxmpp-sdk/license/

[67] http://uniswf.com/

[68] http://www.gnu.org/licenses/lgpl.html

[69] http://en.wikipedia.org/wiki/MIT_License

[70] http://www.ict-prose.eu/

[71] https://github.com/

[72] http://www.opensourceprojects.eu/

[73] http://git-scm.com/

For an open source project with the size of Metafora it is quite difficult for others to improve the software or fix bugs, so the documentation of the software is very important. Prose and GitHub both support wiki pages by providing a DokuWiki[74] like syntax for project documentation. We use this feature to provide high level information about the project and the software architecture and modularization. This should help experienced developers to modify and improve Metafora with little effort. Our source code is mostly self explaining and we documented the more difficult parts with *Javadoc*[75] comments, so developers can generate a full API documentation for Metafora with just one command. We also set up a *Wordpress blog*[76] for Metafora and provide our own DokuWiki on the Metafora demo server[77], which contains the Metafora documentation, Metafora installation instructions and some additional Metafora information.

5.2. Open access

In addition to the research work, which was done during the project, we also want to give teachers and students the opportunity to benefit from Metafora. Therefore we try to make Metafora a well-known platform for teachers and students and give them the possibility to test Metafora. We also want to make it easy for schools to set up and run their own Metafora platform if desired. For the improvement of Metafora, we want to give companies and developers the possibility to further develop Metafora and ground their business on it.

Since our goal is that the Metafora platform is used to improve teaching in schools, we first have to make Metafora a well-known platform for teachers and students. To reach this goal we use multiple platforms to distribute information about Metafora. One of those platforms is our Wordpress blog[78]. Wordpress is the most famous

[74] https://www.dokuwiki.org/dokuwiki

[75] http://en.wikipedia.org/wiki/Javadoc

[76] http://metaforaproject.wordpress.com/

[77] http://metafora-project.info

[78] http://metaforaproject.wordpress.com/

blogging software and https://www.wordpress.com allows users to create a *blog*[79] for free. With the help of the blog, we keep potential users and developers informed about our latest changes and Metafora related activities. We publish Metafora related activities on the Metafora project site, too, but the Wordpress site has very good link rating and offers nice features like posting new blog entries by only sending a mail to a secret mail address. In addition to the Wordpress blog we use GitHub Pages[80], which is part of the Metafora project[81] account we created in GitHub, and the Prose Wiki[82], which is part of the Prose Metafora repository[83], to distribute some Metafora information. Additionally, the advantage of using those platforms is that developers which find the source code also get some information without additional expense.

Because of the limitations of the free services mentioned above and to guarantee some lifetime, we also run our own project infrastructure. The project homepage, which is hosted by Testaluna for at least 5 years after project end, is available at the addresses http://www.metafora-project.eu/ and http://www.metafora-project.org/. This page provides some project information and all public outcomes of the project like deliverables and publications. To be able to provide a Metafora demo instance we also rented a virtual server for the next five years, which is available at http://metafora-project.info and http://metafora-project.de. This server can be used for small experiments and for teachers and students to try Metafora. We also use this server to host our DokuWiki instance which contains the Metafora documentation, a detailed Metafora installation documentation and some extended documentation for Metafora modules on this server.

We already mentioned some possible licenses with their advantages and disadvantages in the section 5.1. We use the *BSD license*[84] for all modules except the Metafora Home application to create the option for companies to improve

[79] http://en.wikipedia.org/wiki/Blog

[80] http://metafora-project.github.io/

[81] https://github.com/metafora-project

[82] http://www.opensourceprojects.eu/p/metafora/wiki/Home/

[83] http://www.opensourceprojects.eu/p/metafora/

[84] http://opensource.org/licenses/BSD-2-Clause

Metafora and build their own products on basis of Metafora. For the embedding Home application we have to use GPLv3 because of GXT, but with less effort the user interface can be rewritten without GXT. The chosen licenses also allow everyone to use Metafora for free and run their own Metafora instance. To make it easier for administrators to set up Metafora, we provide war[85] packets as downloads. Those packets can be deployed as software module on a Java application container[86] like Tomcat[87] directly. We also created and published a step for step install instruction for a full Metafora server, which is part of our DokuWiki documentation. This instruction contains - in addition to installation of the Metafora software – instructions to install and configure all required services, like the used databases, and handles some security aspects like the firewall. In addition to the war files we provide a complete Metafora server, based on *Ubuntu 12.04 LTS*[88] as *VMWare image*[89]. This image can be run with the *VMWare player*[90] locally or be deployed on most of the common virtualization infrastructures and cloud services like *Amazon EC2*[91].

Our Metafora architecture with its open design and strong modularization makes it easy to reuse parts of the Metafora platform. The high level documentation, which describes our Metafora architecture design, is contained in the DokuWiki and helps developers, which want to reuse aspects of the platform, to identify the useful code quite easily. For a detailed documentation of the modules, an API documentation for each Metafora module can be created with Javadoc[92].

[85] http://en.wikipedia.org/wiki/WAR_file_format_(Sun)

[86] http://en.wikipedia.org/wiki/Web_container

[87] http://tomcat.apache.org/

[88] http://www.ubuntu.com/server

[89] http://www.vmware.com/

[90] http://www.vmware.com/products/player/

[91] http://aws.amazon.com/de/ec2/

[92] http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

5.3. How to integrate new tools into Metafora

Metafora is a web-based platform which integrates microworlds and promotes synergy effects between the integrated tools. This section describes how to add a new tool to Metafora and how to interact with the platform. All tools which want to interact with Metafora have to connect to the Metafora XMPP channels, while microworlds with graphical components also needs to be added to the user interface. New microworlds can be integrated with the following steps:

1. Initialise the microworld with the URL parameters
2. Connect the tool to the XMPP channels log user interaction and listen to messages
3. Interact with the platform with command messages
4. Add a new resource card to Metafora, which allows students to use the microworld from a plan
5. Use the extended Metafora features to reference the resource card to an instance, save the state of a mircoworld and restore the state on reopening of the microworld

Analysis components which do not have a graphical user interface, such as PlaTO [7], just have to connect to the XMPP channels and might send feedback messages with different awareness levels if a reaction of the user is required.

5.3.1. Initialise the microworld

Since Metafora is web-based and integrates web-based tools, the first important interaction is the URL and the URL parameters. The Metafora platform extends the URL with a set of parameters to allow automatic user login and interaction with the platform. If a tool is started with a resource card, which is the usual way, Planning Tool adds the URL parameter *ptNodeId*, with the URL encoded[93] id of the Planning Tool card as value, and the parameter *ptMap*, with the URL encoded name of the Planning Tool plan as value. The used localisation is chosen and forwarded with the parameter *locale*. For details about the localisation parameter see the Google documentation[94]. The most important URL parameter is the *token*. The token is a

[93] http://tools.ietf.org/html/rfc3986

[94] http://www.gwtproject.org/doc/latest/DevGuideI18n.html

unique key which identifies a client machine and is part of all interaction XML messages. The parameters *user*, which contains the URL encoded user name, and the parameter *pw*, which contains the MD5[95] hash of the users password can be used to login the user into the opened tool, if the tool requires a login. Tools with an own user account management should also create a new user account if necessary. If there is more than one user the names of the other users are added with numbered *otherUser* parameters. The other parameters are *groupId*, which contains the URL encoded name of the users group, *challengeId*, which contains a unique integer for the selected challenge, and the parameter *challengeName*, which contains the name of the selected challenge.

5.3.2. Connect to the Metafora XMPP channels

For inter-tool communication Metafora uses three multi-user chats[96]. A tool which interacts with the platform has to connect to the channels logger, analysis and command on the used XMPP[97] server. If the tool is Java-based it can use the XMPP Bridge[98], which cares about connection monitoring and reconnection if necessary, but there are also XMPP libraries for nearly all programming languages available. Metafora uses for all interaction messages a special XML format, named *common format*, which structures the information as XML actions. A *document type definition*[99] for that format can be found in the Metafora DokuWiki.

An integrated tool should log all user actions to the logger channel. The content of this channel is saved as a combined Metafora log in a SQL database and is used as input for analysis components.

If a tool has some integrated analysis or log events with high information content, it should post that messages to analysis channel. All integrated analysis components post their results to that channel and the integrated analysis views use this channel to present the context filtered results to the students. The content in the analysis

[95] http://en.wikipedia.org/wiki/MD5

[96] http://xmpp.org/extensions/xep-0045.html

[97] http://en.wikipedia.org/wiki/Xmpp

[98] https://github.com/metafora-project/XMPPBridge

[99] http://en.wikipedia.org/wiki/Document_type_definition

channel should be *indicators* and *landmarks* (cf. D5.2). Indicators are analysis messages with high information content, like selection of parameters for a simulation in a microworld, while landmarks are important steps of the learning process, like success or failure of a simulation. Both message types use the Metafora action XML format. The usage of the landmark type should be reduced to a minimum, because that messages are shown to the students, and if a tool produces too many landmarks useful information can't be perceived clearly in the visualisation.

The Metafora action XML format is quite universal. The action time is like a Unix time stamp, but it uses the current system time as milliseconds. The *actiontype* describes the type of the action. In case of an analysis action the attribute *type* can be *INDICATOR* or *LANDMARK*. Listing 1 shows an example Metafora indicator action. The *classification* can be, for example, create, delete or other. The user tags describe the source and the receiver of the message, if there is one. Tools should add user tags for all users which were involved in an action. The token of the clients should be added as *ip* attribute of the user. We decided to keep the name ip, for the token, because it is quite meaningful, even if it is no real IP address. The *object* element of the message should describe the object which is involved in the action. It can contain object related *properties*. More general information like the users group or the challenge should be included as action properties instead of object properties. This action format should also used for log messages.

```
<action time="1376903785022">
<actiontype classification="other" logged="false" type="INDICATOR"/>
<user id="Nobbi" role="originator"/>
<object id="Nobbi`s logout" type="LOGOUT">
<properties>
<property name="SENDING_TOOL" value="METAFORA_TEST"/>
</properties>
</object>
<content>
<description><![CDATA[Nobbi logged out!]]></description>
<properties>
<property name="INDICATOR_TYPE" value="ACTIVITY"/>
<property name="SENDING_TOOL" value="METAFORA_TEST"/>
<property name="ACTIVITY_TYPE" value="LOGOUT"/>
<property name="GROUP_ID" value="NobbisTeam"/>
<property name="CHALLENGE_ID" value="4"/>
<property name="CHALLENGE_NAME" value="3d Shooter"/>
</properties>
</content>
</action>
```

Listing 1: An example of a Metafora XML action

5.3.3. Metafora tool interaction with command messages

Metafora uses the *command* channel for direct tool interaction. Such interaction can be opening of a new tab on a client or sharing of an learning artefact to the chat or to the discussion environment LASAD. Analysis tools can use command messages to send feedback messages with different awareness levels to students. Metafora supports at the moment the following commands:

- **open tool**: This is implemented with the XML action command of type *DISPLAY_STATE_URL*.
- **referable object**: The command of type *CREATE_REFERABLE_OBJECT* is used to create a referable object in the tool which is defined with the property *RECEIVING_TOOL*.
- **update card URL**: Planning Tool listens to *MODIFY_NODE_URL* commands, which can be used to update the URL of a resource card.
- **feedback**: Analysis tools can use command messages of type *FEEDBACK* with different awareness levels to interrupt the student if necessary.

Listing 2 shows an example of a XML action command for creation of a referable object. The type of such a message is *CREATE_REFERABLE_OBJECT* and the XML object tab needs to have four properties. The property *OBJECT_HOME_TOOL* should be the name of the sending tool. The property *TEXT* should describe the object. It is for example used in chat as object representation. The property *VIEW_URL* should link to a screenshot of the tool or another visual representation of the object and the property *REFERENCE_URL* is the URL of the object. This means, if a student opens the referable object this URL is opened as new tab. It should not include any parameters like token or users. This parameters are added from the Metafora platform automatically. If tool specific parameters are required, like the name of a plan, this parameter have to be added to the URL from the tool. The receiver of the referable object is defined with the property *RECEIVING_TOOL*. To generate a Metafora referable object in chat the value should be *METAFORA* and for creation of a discussion node in LASAD the receiver should be *LASAD*.

```
<action time="1376934942383">
<actiontype classification="create" logged="true"
    type="CREATE_REFERABLE_OBJECT"/>
<user id="tom" ip="ISNOGOODHOME137692683...." role="originator"/>
<object id="0" type="REFERABLE_OBJECT">
<properties>
<property name="OBJECT_HOME_TOOL" value="PLANNING_TOOL"/>
<property name="TEXT" value="report Plan: TomsMap"/>
```

```
<property name="VIEW_URL"
    value="http://metafora.ku-
eichstaett.de/images/processes/report.svg"/>
<property name="REFERENCE_URL"
    value="https://metaforaserver.ku.de/planningtoolsolo/...."/>
</properties>
</object>
<content>
<properties>
<property name="SENDING_TOOL" value="PLANNING_TOOL"/>
<property name="RECEIVING_TOOL" value="METAFORA"/>
<property name="GROUP_ID" value="Awesome"/>
<property name="CHALLENGE_ID" value="1"/>
<property name="CHALLENGE_NAME" value="Demo Challenge"/>
</properties>
</content>
</action>
```

Listing 2: An example of a referable object command

```
<action time="1376935090824">
<actiontype classification="other" logged="true"
    type="DISPLAY_STATE_URL"/>
<user id="tom" ip="ISNOGOODHOME137692683...." role="originator"/>
<object id="0" type="ELEMENT">
<properties>
<property name="NODE_ID" value="TomsMap_node_13"/>
<property name="REFERENCE_URL"
    value="http://test.silentbaystudios.com/...."/>
</properties>
</object>
<content>
<properties>
<property name="RECEIVING_TOOL" value="METAFORA"/>
<property name="SENDING_TOOL" value="PLANNING_TOOL"/>
<property name="GROUP_ID" value="Awesome"/>
<property name="CHALLENGE_ID" value="1"/>
<property name="CHALLENGE_NAME" value="Demo Challenge"/>
</properties>
</content>
</action>
```

Listing 3: An example of an open URL command

5.3.4. Adding a new resource card to Metafora

With the XML commands and the XMPP channels a tool can interact with the platform, but for integration the platform has to know about the tool. The buttons in tools section are hard-coded and a new tool would have to be added in source, but for flexible integration a *resource card* should be used. A resource card is a special type of Planning Tool card which links to a resource. It can be added from the students to a plan and the card can be integrated into the students plan with

connections to other cards. The tool is opened automatically on a client when the state of the card is set to started. To add a new resource card it has to be added to database *metafora*. Figure 21 shows the model for the Planning Tool cards. First a new name record has to be added. The name record only consists of a name id and any unique, unused integer can be chosen as name id. Then a new card description record can be added to the table *iconrelative*. This record contains the name id and has to contain a URL for an image which represents the tool and the URL where the tool can be found. The *category* for resource cards is 8. Cards are only visible if they are translated for the chosen language. The available languages are listed in table *language* and for each language in this table a new translation for the card must be added. Such a translation is a record of table *languagename*, which contains the name id as *name*, the language id as *language* and the localised name of the tool as *word*. New tools should also be added to the *server* table of the *metaforainit* database. If such a named entry in server table exists it can be used into the card definition table *iconrelative*, in angle brackets, instead of the tool URL. This makes it more comfortable to configure Metafora, because all installation specific settings are combined in one place. Now students can add the new resource card to their plans and launch the tool with the help of this card.

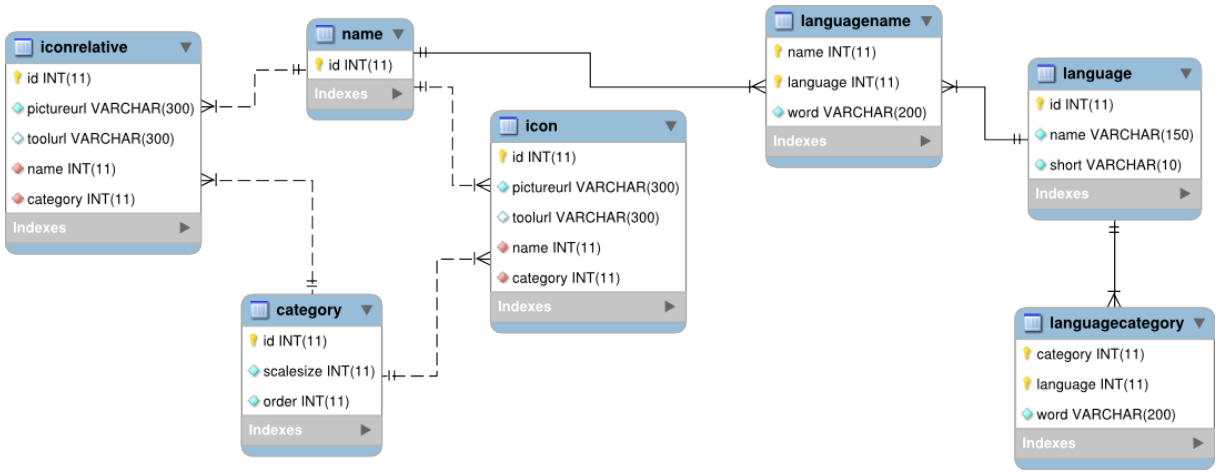

Image 23: Database "metafora": Model for Planning Tool cards

5.3.5. Extended Metafora features

If the tool supports a storable state, it should update the tool URL of the card to a URL which reloads the state of the student. To do this the tool has to send an “update tool URL” command, which is shown in Listing 1. The classification of this message has to be *modify* and the type *MODIFY_NODE_URL*. The card which is

modified is selected with the object element. The attribute *id* of the object has to be the card's id, which is unique for all plans and cards and is added as parameter to the tool URL.

```
<action time="1376938320577">
<actiontype classification="modify" logged="true"
type="MODIFY_NODE_URL"/>
<user id="tom" ip="ISNOGOODHOME1376..." role="originator"/>
<object id="TomsMap_node_14" type="PLANNING_TOOL_NODE">
<properties>
<property name="RESOURCE_URL"
value="https://metaforaserver.ku.de/planningtoolsolo/?planningCard=T
omsOtherMap"/>
<property name="PLANNING_TOOL_MAP" value="TomsMap"/>
<property name="CREATED_PLANNING_TOOL_MAP" value="TomsOtherMap"/>
</properties>
</object>
<content>
<properties>
<property name="RECEIVING_TOOL" value="PLANNING_TOOL"/>
<property name="SENDING_TOOL" value="PLANNING_TOOL_MAP_CREATOR"/>
<property name="GROUP_ID" value="Awesome"/>
<property name="CHALLENGE_ID" value="11"/>
<property name="CHALLENGE_NAME" value="How to feed the world?"/>
</properties>
</content>
</action>
```

Listing 4: An example of an update tool URL command

For reverse linking from a tool to the plan the Planning Tool supports the URL parameter *centerNode*, which should contain a valid, URL encoded card id. Planning Tool highlights this card with a red frame and scrolls it to the center of the view if this parameter is available and contains a valid node of the plan as value.

If a tool wants to use the Metafora file storage or request some old messages it can use the *Metafora Service Module*. Documents can be uploaded to and received from Metafora document store with the help of the *fileupload* servlet, which is implemented in class *DocUploadServlet*. If this servlet receives a HTTP post request it takes the contained multi-part encoded content and the mime-type of the file and puts it to the document store. The file id of the document is part of the response text and requests for file ids can also made with XML action commands. If the post request contains a file id, this document is updated with the new version. The Metafora Service Module also contains a servlet for requesting XML actions with different criteria, like channel

and time. This can be used by analysis components to get messages which are no longer part of the channel history.

6. Summary

This Deliverable described the final system of the Metafora project. It reported about the technical framework system and the integration of all applications used in the Metafora project. Additionally, it presented a tool for designing the visual language of Metafora according to the needs of a learning scenario and a tool for reflection. A major focus of this report was based on the data management, the flexibility of the system and design decisions, implemented interfaces, as well as awareness features and functionality.

The first part of this document dedicated the design decisions of the technical Metafora system. It presented the different modes of awareness, the architecture and security of the framework system and the principles of flexibility and openness of the system.

Afterwards the deliverable presented the tools of the framework system: The main embedded tool called *Home*, the *Planning Tool* which integrates the visual language, a tool for reflection, followed by an application for designing the visual language and the *Workbench* of Metafora.

Subsequently section 4 presented the MeET US study with the purpose to evaluate the user acceptance of the system.

Deliverable 4.2 was concluded with the open source and open access concept of Metafora which was devised in cooperation with WP7 and presented an instrcution how to integrate external tools into the system.

In the attachment you will find the **User Manual** of the Metafora system, which explains how to use all the features and functionalities of the system from the target audience's point of view.

References

[1] Buschmann F., Meunier R., Rohnert H., Sommerlad P. and Stal M. A System of Patterns. John Wiley & Sons, Chichester, 1996.

[2] Harrer A., Irgang T., Lingnau A., Sattes N., Pfahler K. The Metafora design principles for a collaborative, interoperable learning framework. In the 19th International Conference on Collaboration and Technology, CRIWG 2013. Springer, to appear in October 2013.

[3] Hartmann S. Entwicklung eines Konzepts und Realisierung für die Verschlüsselung von Daten innerhalb einer webbasierten kollaborativen Lernumgebung. Bachelor thesis, 2012.

[4] Harrer A., Irgang T., Sattes N., Pfahler K. SoCCR – optimistic concurrency control for the web-based collaborative framework Metafora. Proceedings of the 18th International Conference on Collaboration and Technology, CRIWG 2012, Springer, pp. 153- 160, Raesfeld, Germany.

[5] Irgang T., Sattes N., Pfahler K., Lingnau A., Harrer A. Reflektionsunterstützung im Metafora System durch eingebettete Learning Analytics Werkzeuge. Accepted to the Workshop Learning Analytics at the DeLFI 2013, 8th of September 2013, Bremen, Germany.

[6] Fuchs L., Pankoke-Babatz U. and Prinz W. Supporting cooperative awareness with local event mechanisms: The GroupDesk system. In Hans Marmolin, Yngwe Sundblad and Kjeld Schmidt: Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work, ECSCW ’95.

[7] Harrer A., Herbst V. PlaTO – the planning tool observer in the multi-tool ELE Metafora. In ITS Workshop Proceedings on Intelligent Support for Exploratory Environments: Exploring, Collaborating, and Learning Together, 2012.

[8] Wise, A.F., Zhao, Y., and Hausknecht, S.N. Learning analytics for online discussions: a pedagogical model for intervention with embedded and extracted analytics. In LAK 2013 - Learning Analytics and Knowledge, pages 48-56, 2013.

Further reading (on specific aspects)

Harrer A. Analytics of collaborative planning in Metafora – architecture, data, and analytics methods. Proceedings of the third Conference on Learning Analytics and Knowledge, LAK 2013.

Harrer A., Pfahler K., de Groot R., Abdu R. Research on collaborative planning and reflection – methods and tools in the Metafora Project. In EC-TEL Eight European Conference on Technology Enhanced Learning 2013. Springer, to appear in September 2013.

Harrer A., Pfahler K., Lingnau A. Planning for life – educate students to plan. In ICALT the 13th IEEE International Conference on Advanced Learning Technologies 2013. Springer, to appear in July 2013.

Harrer A., Pfahler K., Lingnau A., Herbst V., Sattes N., Irgang T. Kollaboratives Planen und Lernen mit der web-basierten Lernplattform Metafora. In DeLFI 2013 Mensch & Computer. Springer, to appear in September 2013.

Herbst V., Harrer A. Ermittlung von Kollaboration und take-up in Lernszenarien durch autonome Agenten. Accepted to the Workshop Learning Analytics at the DeLFI 2013, 8th of September 2013, Bremen, Germany.

Lingnau A., Harrer A., Pfahler K. Bringing learning science and user interface design into one loop of joint research interests – the SoCCR case in Metafora. Learning Science and Human Computer Interaction Workshop at the 10th International Conference on Computer Supported Collaborative Learning. CSLC 2013, 15th June 2013, Madison, Wisconsin, USA.

Harrer A., Schmidt A. Blockmodelling and role analysis in multi-relational networks. Social Networking Analysis and Mining journal, Springer-Verlag, Wien 2013, Online ISSN 1869-5469, Journal no. 13278.