DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS
BIRKBECK COLLEGE
UNIVERSITY OF LONDON

Development of Web-based TA Tools for the MiGen Project

Sokratis Karkalas

29/4/2013

Table of Contents

Abstract

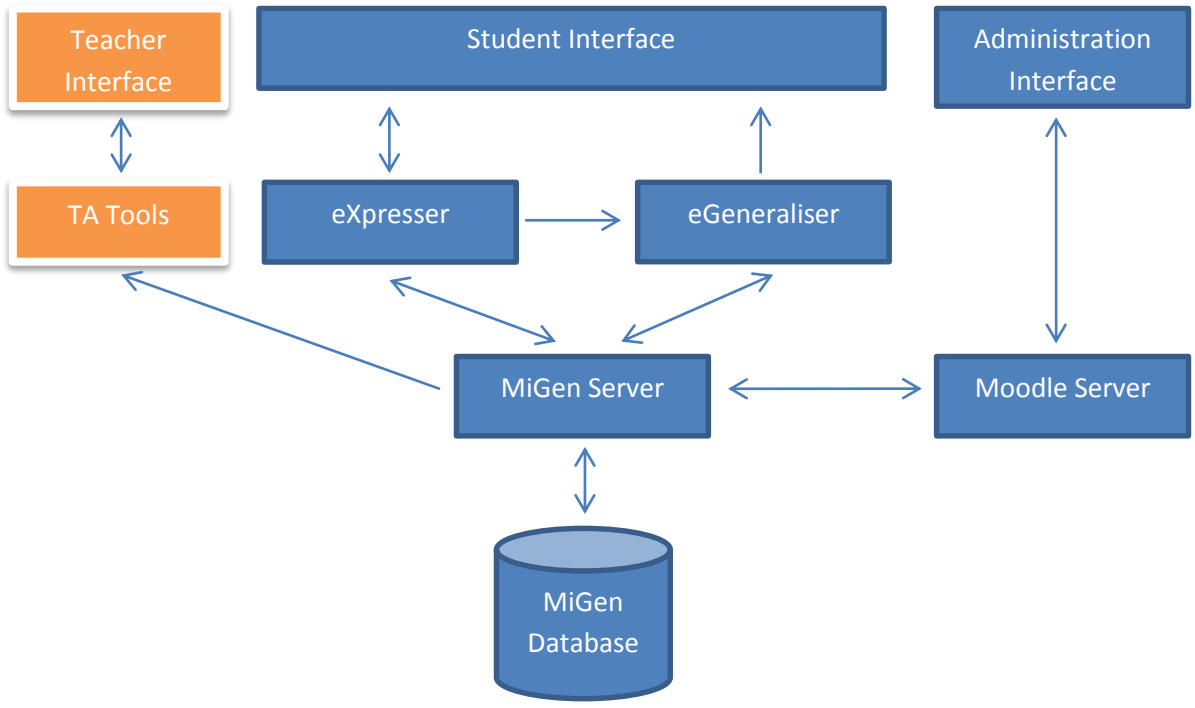
This report presents the design and implementation of a web-based system intended to extend the teaching assistance support of the MiGen Project. The original implementation of the MiGen project provides a set of TA tools that are developed in Java and were intended to be used in a classic two tier client-server setting. This extension implements a subset of these tools, namely the classroom dynamics tool and the goal achievement tool using, an n-tier web based architecture. This project was supervised by Dr. Sergio Santos and Dr. Manolis Mavrikis, both members of the LKL.

Introduction

1. System Design

As said above, the initial development of this system was implemented in Java using a two-tier, client-server architecture. The data at the back end was held in a JavaDB database [refs to ECTEL paper & IEEE TLT paper]. The eXpresser tool and the intelligent support component of this system have already migrated to an n-tier web-based scheme. The new system utilises the Google's AppEngine for storing the data, business components in the middle tier and client-side components for interfacing with the users.

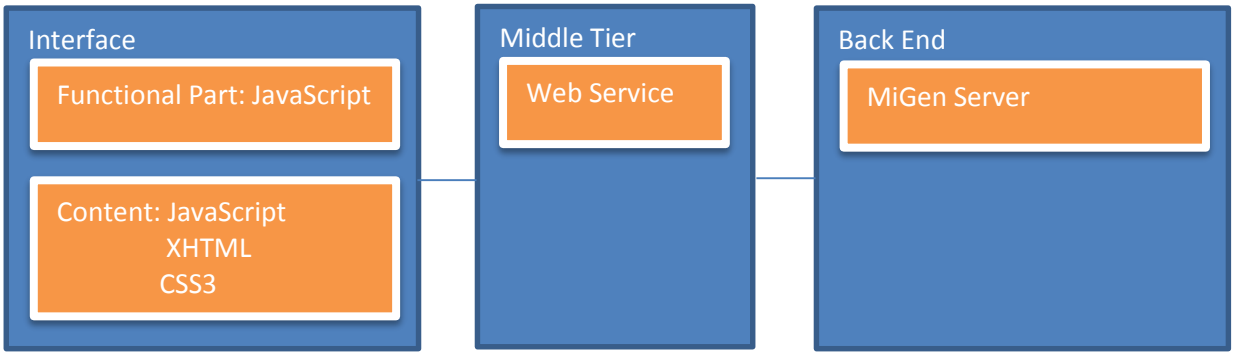
The new system architecture is depicted in the following diagram:



The contribution of this project is the part coloured in orange.

1.1 Architecture of the TA Tools

The three main tiers of the TA Tools are presented in the following figure:

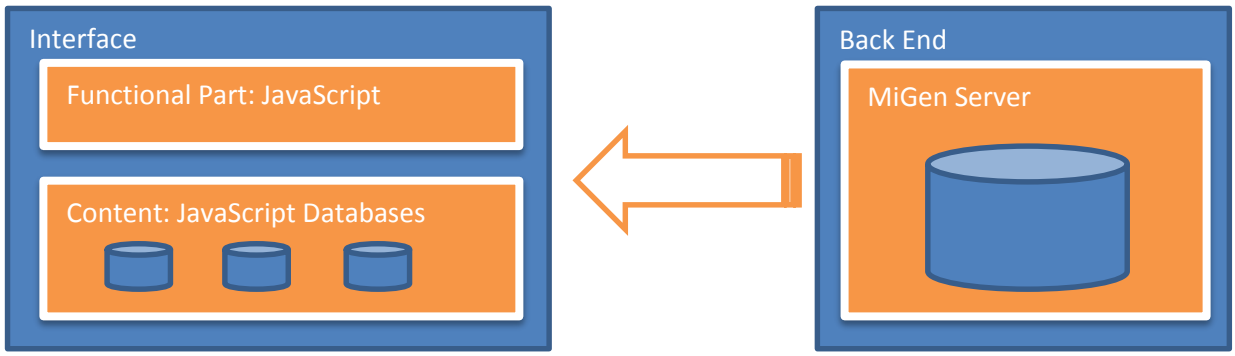


This architecture is suitable for relatively low volume sites. It utilises a relatively fat client at the front end and there are no business components in the middle tier. This arrangement may imply performance and scalability issues if the demand of this service is high. No processing takes place in the middle tier. The whole bulk of the processing takes place in the client. The functional part is responsible for the interaction with the user and the retrieval of data from the MiGen Server via the web service in the middle tier. The net effect looks like a two-tier client-server configuration.

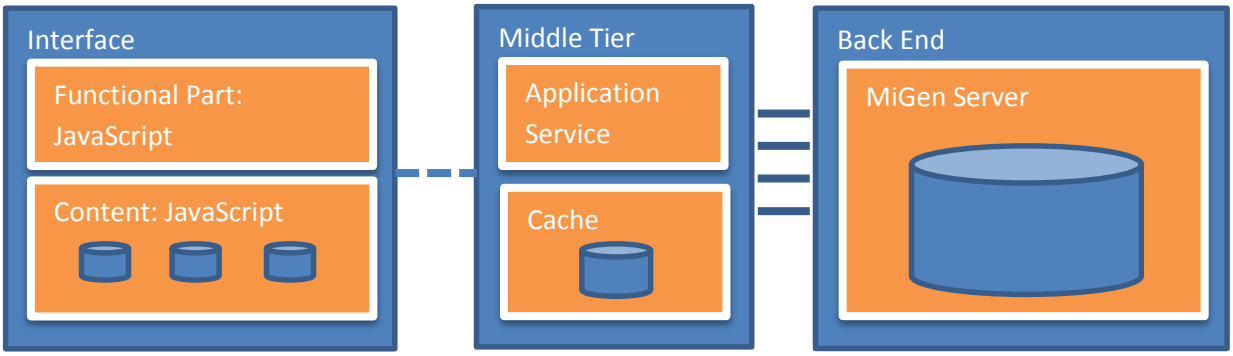
1.2 Important Design Issues

One of the main considerations in web based applications is efficiency of the system in terms of response time. There are two factors that affect performance most. The first is the distribution of workload and the second is network traffic. These two factors are interdependent to an extent. It is safe to assume that network traffic should always be as minimal as possible. The proper distribution of workload between the tiers can have a significant impact in network traffic, therefore workload distribution can be thought of as a determinant. It determines performance transitively, through network traffic, and directly through service availability. If, for example, the client component does not perform any processing at all, then it is inevitable that the network will be heavily used for transferring the results of the processing that takes place on the server. Therefore, a good strategy to reduce network traffic is to do some data caching and perform some processing on the client. This justifies the transitive dependency. If there is a component that is very heavily loaded it might become less responsive, if not unusable. If, for instance, this is a data intensive application and there are a large number of Internet users accessing the database in parallel, the DBMS may become overloaded. This is an unfortunate situation that may lead to a denial of service through a cyclic process. If users are denied the service, they usually try harder requesting the same thing multiple times causing the service to reserve even more valuable resources for handling them.

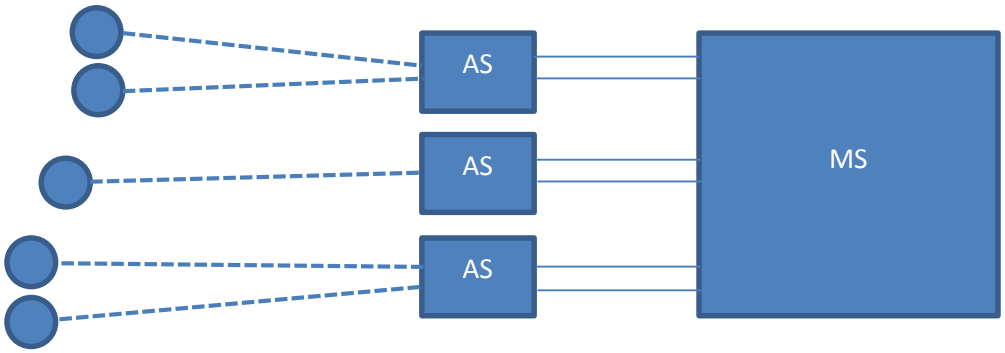
In this system the data is given by the MiGen server in a single file. This file is sent directly to the client where it is processed and used to form a local (client-side) database (JavaScript content). The local database is updated in an incremental fashion when new data arrives, even if the new data contains information that is already stored locally. The client always uses whatever data is available in its local storage and never queries the server again for the same data. Therefore, caching the data at the client is used for minimising network traffic.



Performance from that point of view may be good but there are two obstacles in this process. The first obstacle is that the database is directly accessed by clients. This is not appropriate for web-based applications. If the demand for the service is very high the database server may become overloaded. Even if the database service is based on a clustered configuration that offers load balancing and limitless availability this is not a proper utilisation of computer resources and it will most likely incur the system with extra cost. A reasonable arrangement in this case would be an application service in the middle tier to cache client requests and utilise a pool of reusable permanent connections to the database. That would reduce database usage to a minimum, thus minimising the cost, and would improve performance.

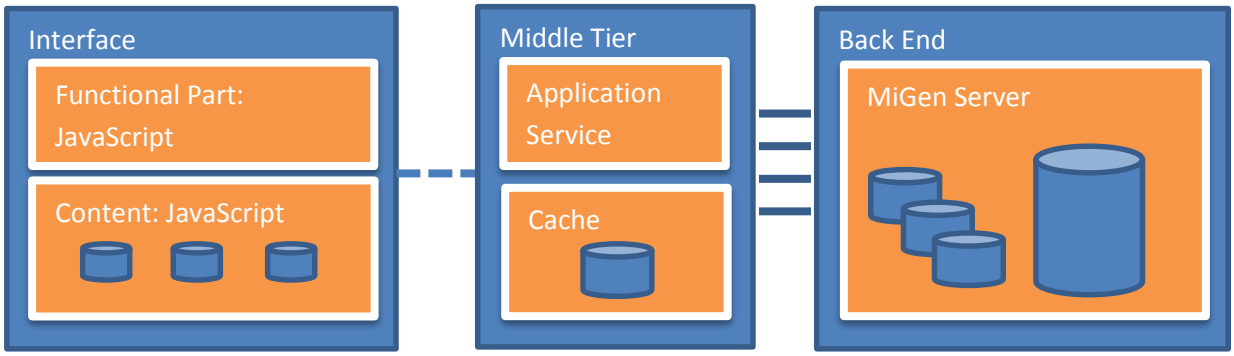


One derivative advantage in this case would be the ability to scale better without sacrificing performance by cloning the application service and maintaining the database usage to a minimum. The system in this case would also be more modular, easier maintainable and extensible.

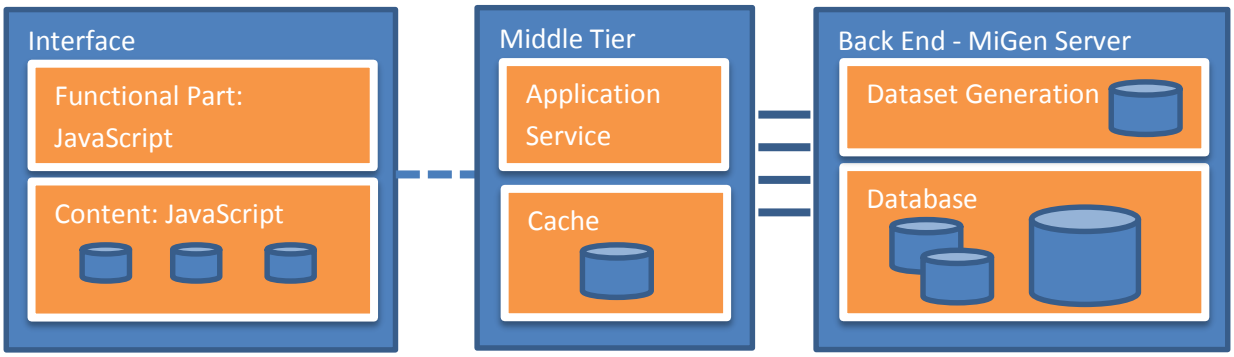


One other obstacle is the fact that all the information generated by the server, relevant or not to what the client needs, is always contained in the file. That implies that the system exchanges information that may not be needed and not consumed at the client side. This is an inherent inefficiency of the system that relates to the decision to use a standardised common XML format to encode the data. Standards must exist and should always be used, so the question is not about using the standard but about how the standard must evolve in order to provide enough flexibility in case the consumer of the information needs only a portion of the data and not the whole set.

The solution to this problem is three-fold. The first part presupposes a revision of the common format so that it can be more flexible and permit partial representation of the data.



The second part is related to where the processing of this data should take place. Since this is a data intensive application the initial processing of the data should take place at the database server side. If there is a specific set of data that is required by the middle tier then the formation of this data must take place on the database server and only the resulting set sent to the consumer.



The third part is related to the data representation used for the package to be transferred over the network. The use of XML is pretty standard for this type of communication but there is another standard that emerges in new developments. The JSON format is simpler, lighter and directly usable in JavaScript. That means that if the application server at the middle tier needs to consume a dataset retrieved from the database server in its JavaScript local cache, no parsing or extra processing will be

required. The dataset can be consumed directly and form JavaScript objects. This is a great benefit in terms of resources utilisation and complexity of code.

2. System Implementation

2.1 The Database Management System

The MiGen server utilises a native object-oriented DBMS. This is the Google AppEngine. It is a system that resides on the cloud and promises good performance through load balancing and limitless availability through on-demand dynamic allocation of more processing power and memory space within a cluster or servers. The amount of data that can be exchanged and the number or requests that this system can satisfy is theoretically limitless. In practical terms there is a threshold after which Google starts charging the services provided. The obvious advantage of using this system is that it is free (if it is used sensibly) and it does not entail any administrative costs. It offers a rich and programmable environment through Java, Python and Go. It can be enhanced with web services and provide any type of server side processing needed.

2.2 The Middle Tier – Web Service

In this part of the implementation another popular service that belongs to the world of cloud computing is used. This is the Dropbox system. This system is not intended to be used as a web server, but it proved to be very reliable, fast and convenient in terms of administration, data security and data portability. Unfortunately it cannot be used for anything more than that because there is no support for the deployment of business components.

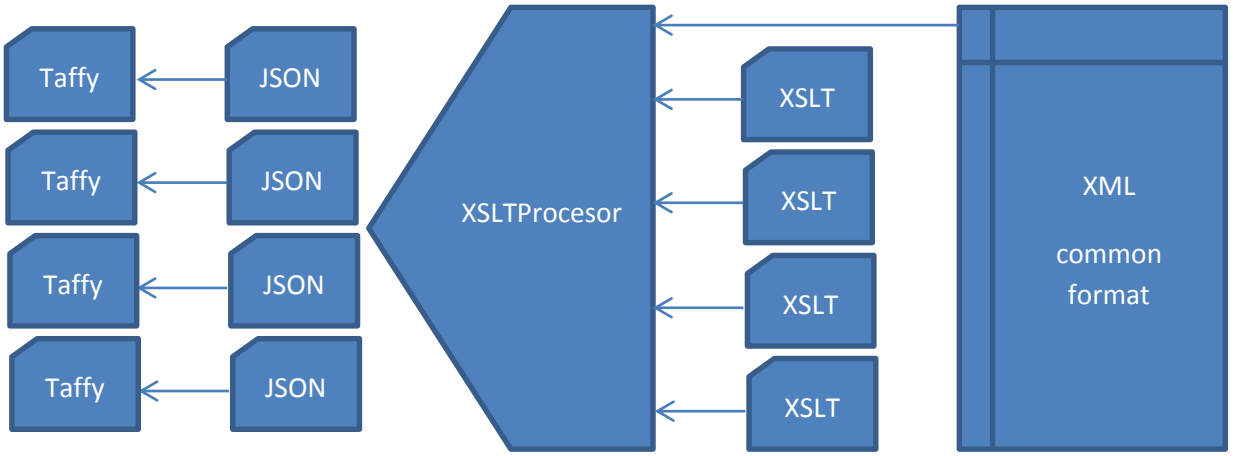
2.3 The Front Tier – Client Service

As shown in the above graphical representation of the system, the client side is relatively fat and incorporates two parts. The first part consists of the usual components of a typical web page. The content is written in XHTML and CSS3 stylesheets have been used for managing the layout. The content is also enhanced with JavaScript databases that cache information about the following entities:

- Tasks given to students
- Users (students)
- Users' state
- Users' goal achievements

A separate database is maintained for each one of them. The TaffyDB JavaScript library has been used for implementing the databases. This proved to be a good decision since the library has a very small memory footprint (lightweight), it provides enough functionality for basic DDL and DML operations and it is relatively easy to use. Furthermore, it supports direct instantiation of JSON objects, it performs very well and it also is bug free and reliable.

The databases are populated automatically using JSON. The information that is retrieved from the MiGen server is not in JSON format. The only format supported by the MiGen server for the time being is XML. Information regarding all the above entities is packaged in a single XML file according the common format standard […] and sent to the client on demand. This file is then processed on the client, analysed and decomposed into separate JSON files that correspond to the entities of the above list. The processing is performed through XSLT files and the corresponding processor that is built-into the browser. The process is depicted in the following diagram:



This system is designed in such a way so that if the data retrieved overlap with the data already stored in Taffy, then the system keeps only the new data and discards the existing part. That ensures that the database updates are always incremental and thus we don't have duplicate values or perform unnecessary processing at the (client) database side.

The request sent by the client for the XML file is always asynchronous so that there is no need to refresh the whole page every time data needs to be transferred from the server. Another consequence of that is that the changes are barely noticeable by the user which enhances the user experience. The most important benefit though is the fact that there is no need to maintain state of the user interface between the calls. Since the page doesn't refresh the graphical objects on the page remain in their current positions and there is no need to maintain state information in order to redraw them in the correct place. The technology that has been used for these calls is Ajax. The advantages of using it in terms of performance are self-evident. If we had to refresh, we would have to recreate everything on the page, including the databases. The updates wouldn't be incremental in this case.

Another technology that has been used at the client side is JQuery. This library is not standard JavaScript but it has been extensively used in the industry and it is considered to be a fairly safe option for enhancing the functionality of the page without extensive programming effort. It provides a well-designed framework within which one can perform asynchronous network operations, manipulate page elements with ease and provide sophisticated interactive visualisations. The visuals of the interface have been developed with JQuery.

2.4 Other Issues

There are a number of issues of secondary importance that have to be mentioned in this report. The Moodle system [more info++] is not tightly integrated with the rest of the MiGen system and as a consequence the file generated regarding the context information cannot be retrieved as part of the file that contains the rest of the data. That means that this file requires separate processing by the tools. The same happens with the information regarding tasks and goals. For the time being this information is received through separate static files from different domains. The fact that this information is not organised under a common standardised format means extra administrative overhead and extra processing overhead from the TA application. Furthermore, the fact that these files are accessed asynchronously through Ajax introduces another problem. Cross domain scripting is not allowed in the Ajax world unless it is explicitly permitted by the sites involved. That entails even more administrative overhead.

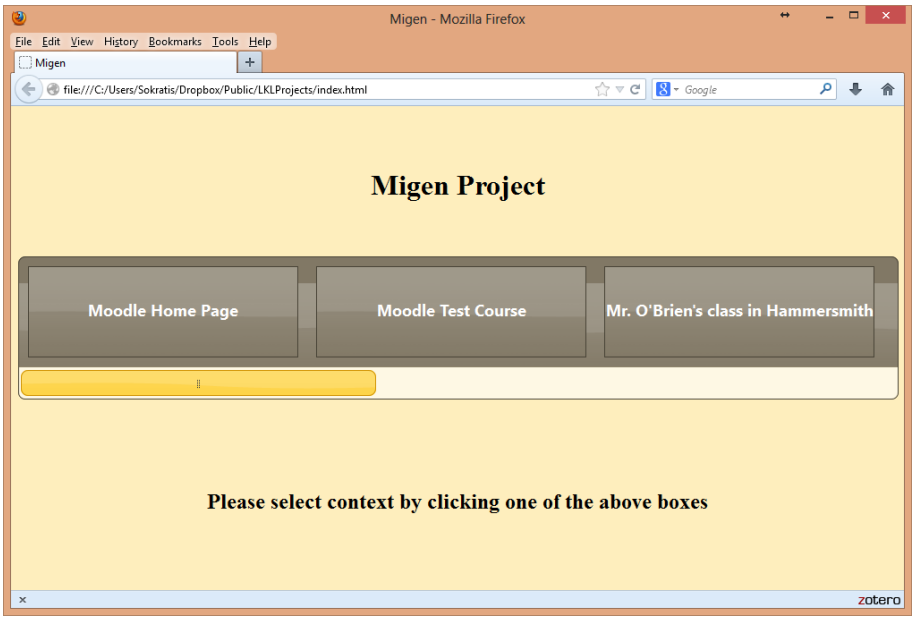
Another issue that was raised during the implementation was the incompatibilities between the official DTD that defines the common format and the structure of the actual file generated by the MiGen server. Fortunately these issues were resolved in the process. After all the common format needs to be revised anyway.

The system in its current state can retrieve information from the MiGen server either automatically or after manual request by the operator. In any case every new request is automatically parameterised so that no data duplication occurs between subsequent requests. The consequence of this is that network traffic is kept to a minimum. Even if there was the possibility of some data duplication the client databases are configured so that only incremental update is possible. That minimises the overhead of processing the same data multiple times at the client side. The manual method implies that the TA tools may not be updated as frequently as needed to always reflect the current state of the classroom. If this method proves to be inadequate the system can also be set to use automatic updates. This essentially means polling the server for new data at certain time intervals. The problem with this solution is that the client will possibly create unnecessary network traffic that would increase the workload of the server (for handling these requests) if there is no new data available. A far better solution would be to use the Google channel client API. This is a JavaScript-based application that can be executed at the client side and communicate with its counterpart on the server through a permanent channel. This channel can provide information that new data is available directly from the source almost immediately after this data is generated. The problem is that the TA tools need to be hosted on the Google-based MiGen server to take advantage of this functionality. In the present time the tools are hosted in Dropbox which is a cloud-based primitive web service not able to take advantage of this functionality.

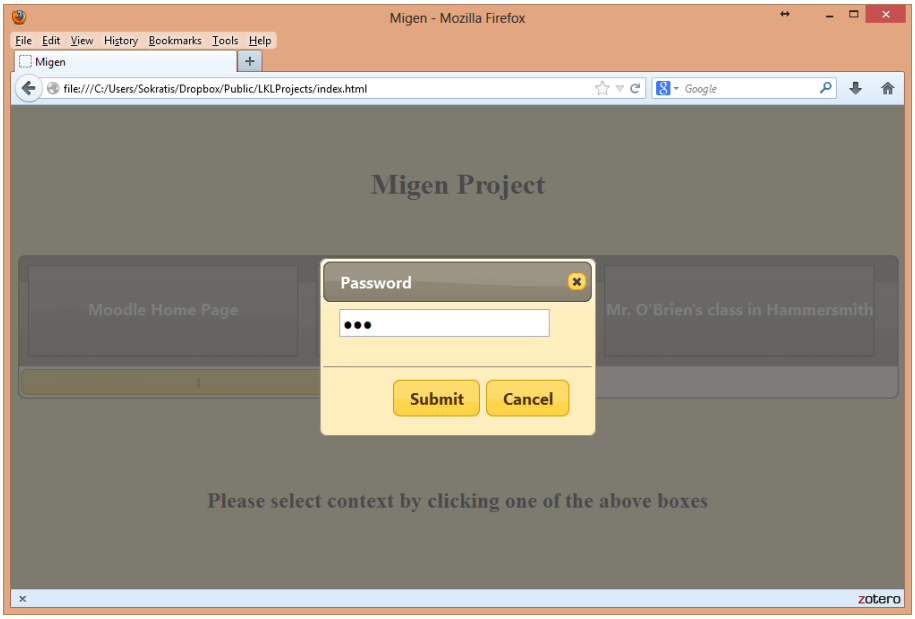
A final issue related to the implementation of this project is the design of the code. JavaScript is an object-oriented language and it makes sense to be used as one. Most of the code written for this project is organised in a procedural manner and therefore it is not very modular and easily manageable. This decision affects many aspects of the system like maintainability, extensibility and scalability. If the system is modular then new components added to the system can provide more functionality without compromising the availability of existing components.

3. System Presentation

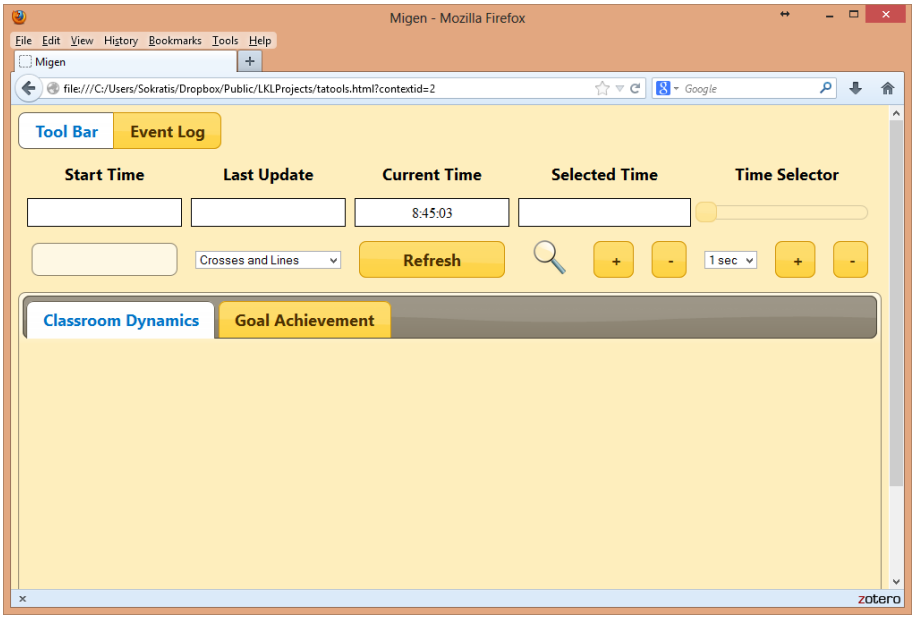
The front page of the system presents a series of courses created in advance by the system administrator from Moodle. Each one of these courses corresponds to a specific class of a school that takes part in the program. Such a course is also referred to a context.



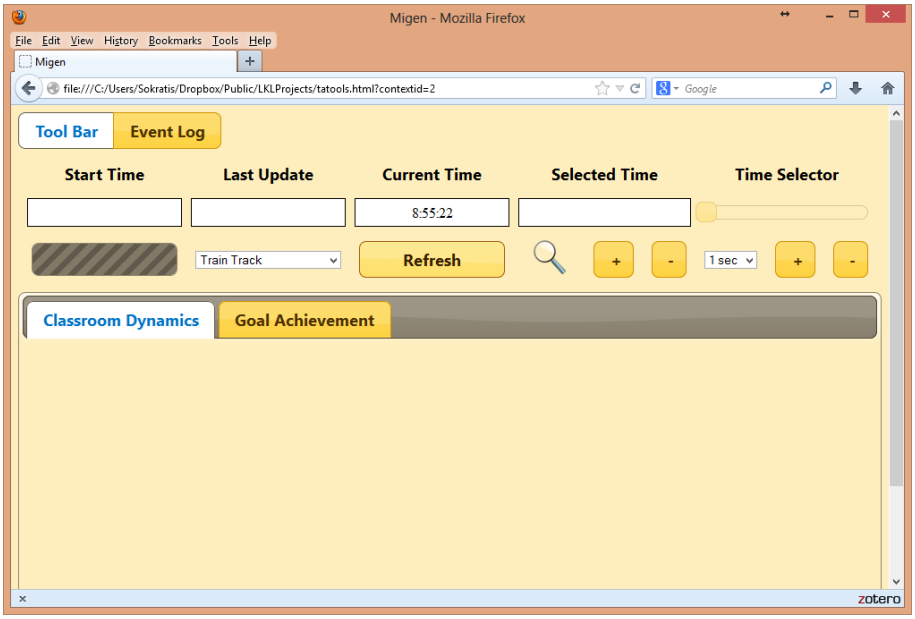
Access to these courses is password-protected. The user is prompted to select the course and provide the password through the following dialog window:



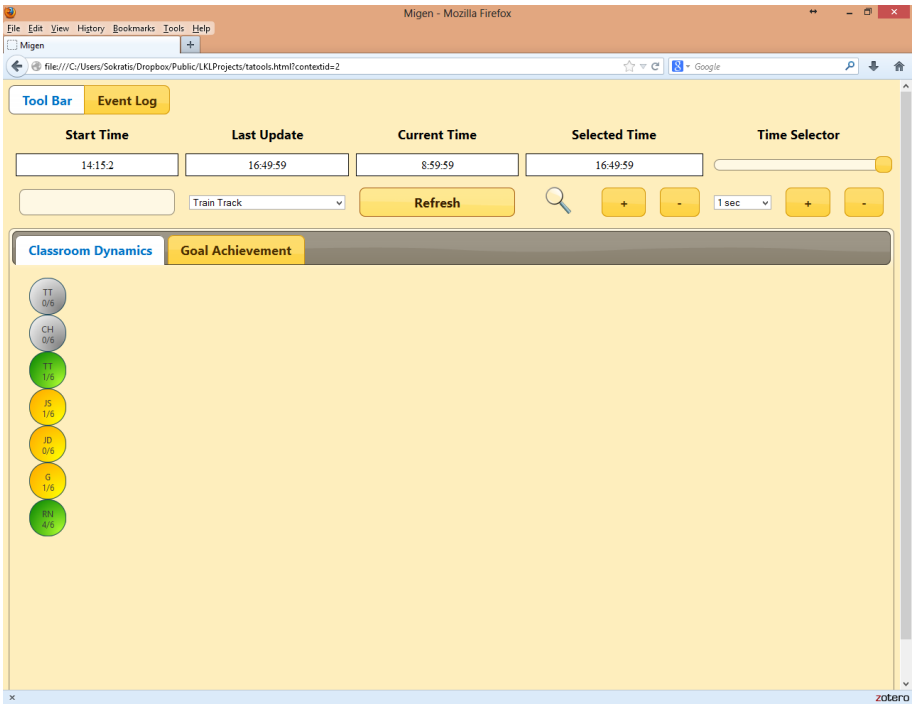
If the submission is successful, the user gets redirected to the main page of the tools.



The first thing to do at that stage is to select the task from the list of available tasks that have been set for this particular course. After that a click on the button 'Refresh' will issue an asynchronous call to the MiGen server to retrieve indicators about the users' actions in the virtual classroom. The only visual indication that this process takes place is the graphical progress bar on the left hand side of the window:



As soon as the data arrives, the system updates its local storage (JavaScript databases) and then queries the databases to display the required visualisations.



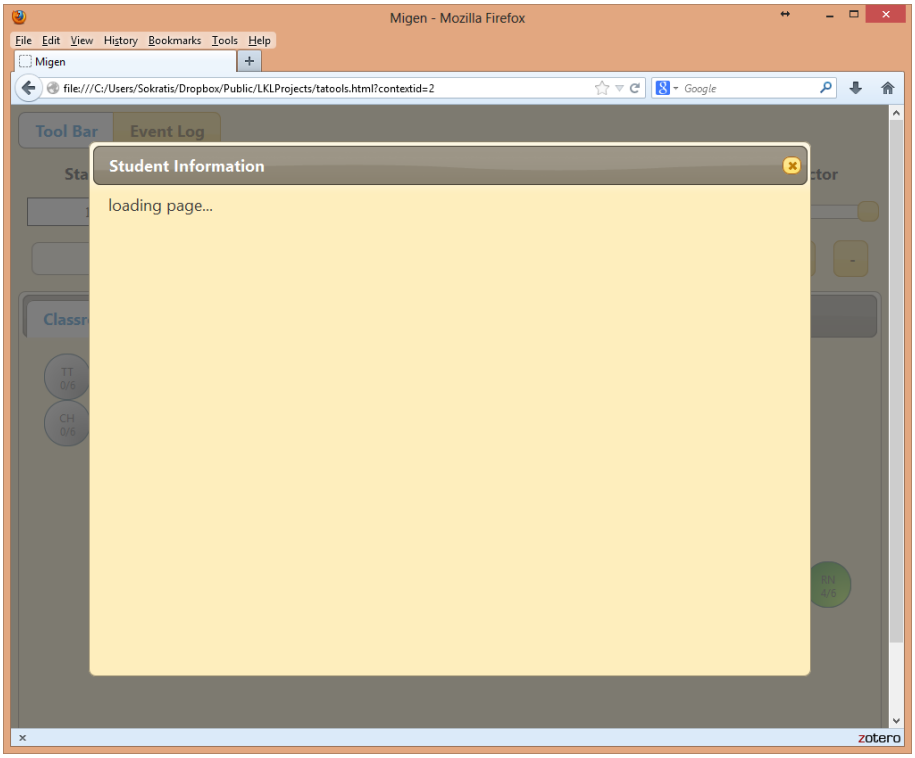
The main page is divided into two parts (tabs). The first is the 'Classroom Dynamics' that present the spatial configuration of the classroom. Every circle in this area corresponds to a student and her physical location in the classroom. The circles are draggable so that the operator can place them to their correct location using the mouse.

The different colours in the circles indicate the current state of the student. The colour scheme used follows:

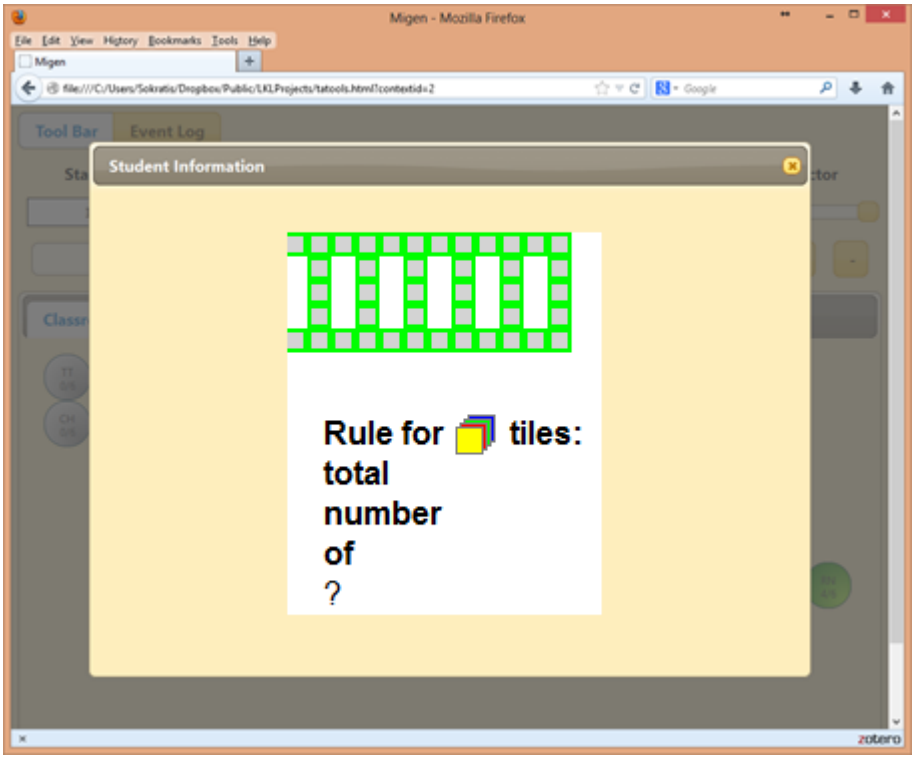
- Green: the student is actively engaging with the task given.
- Amber: the student is not active (for some period of time no activity has been recorded).
- Red: the student is struggling with the task (unsuccessfully so far) and is waiting for assistance.
- Grey: no action indicators have been received for this student so far.

The text within the circles provides a short reference to the students' names (identifiers) and the number of goals achieved so far. If the operator double clicks on a circle then an applet showing the current user activity gets displayed within a modal dialog window. This applet is not a static screenshot. It offers interactivity and the operator can actually intervene to what the student is doing. This applet is relatively demanding in terms of memory footprint, processing power and network activity hence the decision about not permitting simultaneous displays for many students.

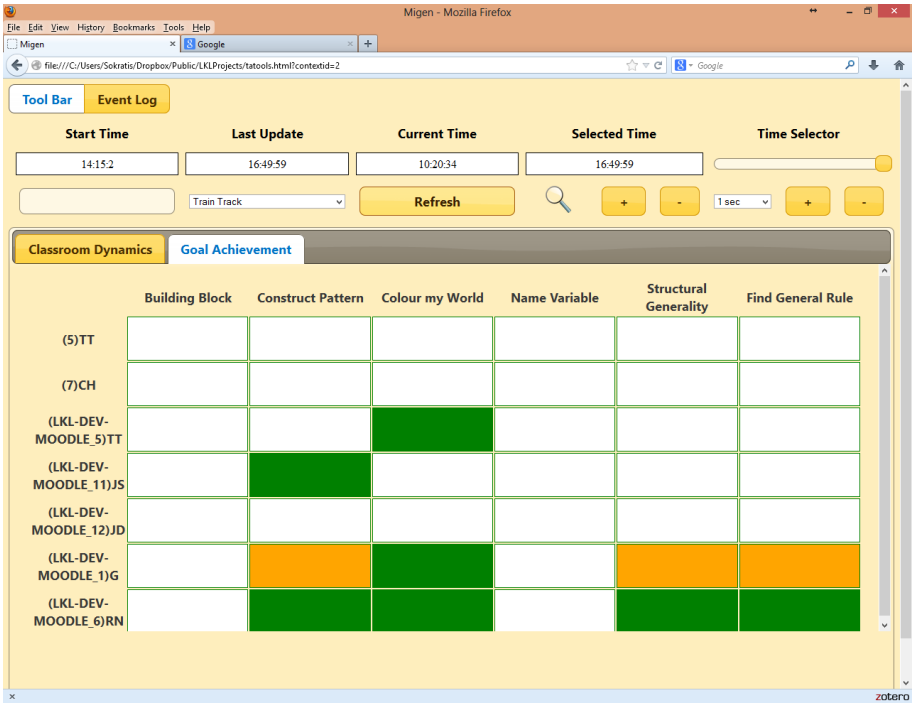
In the present implementation the cross-domain scripting constraints of Ajax did not allow the utilisation of this applet. If a double click occurs the following screen gets displayed:



Under normal circumstances the window would look like the following screenshot:



The second part of the page is the 'Goal Achievement' tab. This page displays in a 2-D matrix the goals achieved so far by the students. The columns of this table represent the goals set for the selected task and the rows represent the students. The intersection between a column and a row gives information about the particular goal achievement by a student.



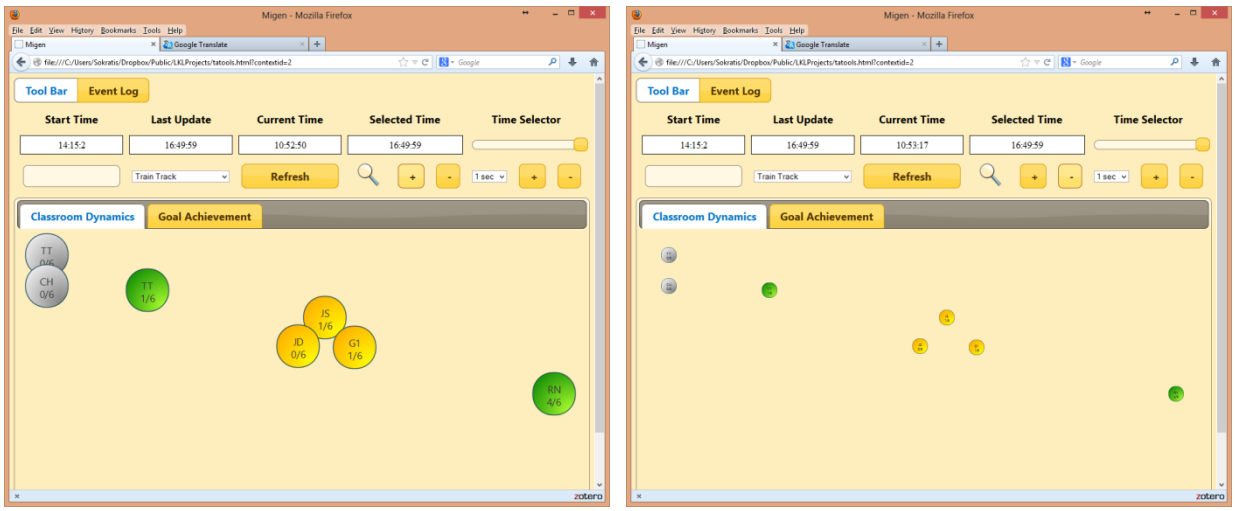
Again, there is a colour scheme employed to encode that information.

- White: the goal has not been achieved.
- Green: the goal has been achieved.
- Orange: the goal was achieved but the user is making another attempt.

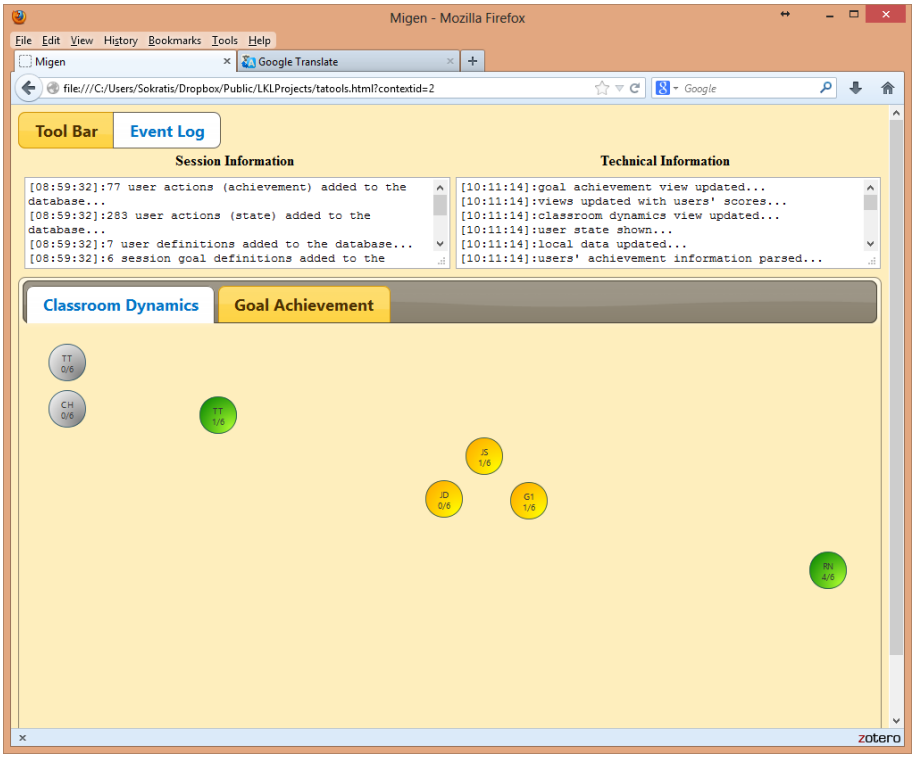
Above the main presentation window there is a series of controls that can either provide more information about the current session or used to configure it according to the needs of the occasion.

At the top level there is a series of text fields that present information about the time. The information given is the time the session started (the first activity indicators recorded), the time the most recent indicators were received (last update), the current time (the time generated by the client system) and the time selected by the user through the time selector slider control. The user can use the slider control on the right hand side of the page to go back and forth in time within the time period recorded in the local databases. That automatically changes the visuals on the screen to reflect the state of things that correspond to the selected time. This is a feature that provides a great help to the instructor that may want to inspect previous (historical) data. Time can be adjusted by the slider, or if a finer control is required, by the combination of the controls below the slider.

Another useful feature is the ability to change the focus of the main presentation page. This can be really handy in situations where the class comprises many students that the dimensions of the screen cannot accommodate. These settings are only applied on the currently visible portion of the window.



The set of buttons at the very top of the page can be used to change the upper part of the page (the part that contains the controls) with an event log that presents useful information about the session and the technical side of it. This information is really useful in tracing regarding the smooth operation of the application and debugging in general.



4. Conclusions – Suggestions for Further Improvement

The system works and it is fully functional therefore the initial objective is fulfilled. The very few things that don't work are related to insignificant details that in any way cannot reduce the value of the outcome.

The system has not been tested extensively and thoroughly. A proper test requires a reasonable number of students in a real classroom working in a typical school setting. The fact that the system seems to perform adequately fast with data generated by one or two pseudo-users is not a safe indication of its performance and thus its usefulness. A test with a large number of concurrent classes would be a good test not only for the client-side TA tools but also for the data server at the back end. It would be interesting to see the rate at which the workload of the server and the network traffic increases in relation to the concurrent number of users accessing the service. If such a test was possible we would be able to identify the threshold after which it would be sensible to change the architecture of the TA tools and to make them more scalable with less cost as suggested in 1.2.

A very beneficial and relatively inexpensive addition to the system would be to utilise the Google channel client utility. That would make the tools more accurate and responsive with less human intervention. That of course requires the migration of the tools to the MiGen server.

A final and very important issue that needs to be considered carefully is the revision of the common format. This format should be general enough to encode all the information required in every different context and flexible enough to permit the formation of individual parts separately. In a sense that would mean the decomposition of the common format into a collection of schemata to

represent the individual aspects of such a system. In practical terms it would be a wise decision to start using XML schemata instead of a DTD for the definition of this data.

References

.

.

Appendixes

A – The common format