SFIDE: challenges towards synchronous interaction in e-learning

Michele Albrigo (michele.albrigo@univr.it), Roberto Burro (roberto@psico.univr.it), Olga Forlani (olga.forlani@univr.it), Franco Bersani (franco.bersani@univr.it), Corrado Ferreri (corrado.ferreri@univr.it), Giovanni Michele Bianco (giovanni.bianco@univr.it)

University of Verona - Italy

Abstract - History of the development of a synchronous tool for e-learning, with University of Verona in the unusual role of services provider.

I. Introduction

Recently, the University of Verona has been deploying several initiatives that involve e-learning methodologies. Almost all of them are related to internal courses that, at different levels, apply remote and synchronous/asynchronous learning. We discussed some features of those e-learning tools in other papers [1,2].

Starting from mid 2006, our University has been experiencing an interesting and challenging e-learning initiative that doesn't involve classical academic courses. The initiative is aimed to provide learning courses deployed in the whole regional area, whose goal is training bricklayers. This appeared for us as a real challenge for several reason: it involves at the same time the seven cities of the Veneto region, classes are located in high schools, teachers often do their activity from their private offices, bricklayers aren't very familiar with PCs and software in general, not only with e-learning tools. Teachers work in presence, so synchronous tools and video streaming were needed, but they also asked for feedback tools, so asynchronous methodologies needed to be implemented, we had to provide this service to train over 1200 bricklayers and, finally, this had to be an inexpensive project, with less than a 50.000 euro budget.

At the same time, this project was very exciting: the University could open its "boundaries" to the territory, thus facing the major criticism in Italy, i.e. the "auto referentiality" of the academic world. As a second point, we had to work with different ISPs, different QoS, different speeds and different ACLs, since the schools weren't required to use some kind of dedicated connectivity for the project.

For these and other reasons, this initiative has been named "SFIDE", an acronym that stands for "Sviluppo della Formazione Individuale a Distanza in Edilizia" (development of individual remote learning for building trade), but the italian word "sfide" also means "challenges".

Our board were impressed about our optimistic view: they kept asking us whether we were aware of the risks. Nothing existed, and exists at the moment in Italy, that could be compared to the SFIDE project, so we could not take advantage of other experiences. We only were confident on our e-learning framework: Moodle.

This paper tells the SFIDE project "story". SFIDE is still working, and we are asked to enlarge its scope to other activities.

II. Initial analysis

The initial requirements were based on the contractor's experience on a similar platform during the previous year. They expected to be able to show lessons on a shared blackboard, where users could interact easily with basic instruments like a drawing pen and text boxes. Users should also be able to interact with the teacher (with an audio/video channel) and between them, with a text chat. This interface would have been used for two different kind of classes, a low-level one, with no application sharing involved, and a higher level one, which would have used application sharing to allow teachers, tutors and students to interact over an AutoCAD session.

After a first analysis of the available software, we decided to implement our own solution: this allowed us more flexibility over the contractor's requirements, and posed no strict limits on the number of clients (which is very important to extend the live-class usage to other courses and projects).

A. Requested features

During the development of the application, a number of other feature requests emerged. To allow interaction between the teacher and the tutors (and the help-desk operator), a second chat channel has been created, separated from the main one, and not shown to students. Group of students are now monitored, in order to know how many of them are connected from each site, and to know from which site each student connects. Teachers are now able to send pop-ups to single students or to the whole class. Teachers and tutors also wanted to avoid multiple connections from a single account, resulting in some users being listed two times in the live-class, so connections had to have an inactivity timeout, some keep-alive dialogue and an upper limit on number.

The Third Advanced International Conference on Telecommunications (AICT'07)
0-7695-2843-0/07 $20.00

B. Environment requirements

The live class itself is integrated in a widespread LCMS, the open source project Moodle. At first, the interaction between the live-class and Moodle is quite loose, this poses no problems at all on the server side. The live-class interaction is entirely browser based, so, on the client side, requirements are quite general: a browser with Flash support, an audio output device, and a web-cam (otherwise no video interaction will be possible). On the server side, since interaction between the live-class and the Moodle server is really low, there are no specific requirements. Our setup consists of two physical servers, one for Moodle and one for the Flash Media Server. Once their requirements are met, there are no other specific requirements. The live-class will be seen as a resource in a Moodle course, blending it with the course in which it's contained. There's no real perception of the two different environments from a user's point of view.

III. Development

A. Initial analysis

Any analysis for the development of the synchronous module, the live-class, had usability and platform independence of the final product as its main target. The combination of Flash Media Server (FMS2, a server platform to create multimedia applications and audio/video streaming to and from flash clients) with ActionScript (AC, an ECMAS syntax scripting language for flash player applications' development) seemed to be one of the possible choices to implement what we've been asked. Distributing a Flash platform allows a consistent experience between different browsing platforms and an easy integration with any other web interface. Having the same possibilities of a stand-alone application while being entirely online largely justifies the choices we made.

B. Two programming levels

FMS2 bounds the development on two different levels: client-side and server-side. Some functionalities are allowed only in one of the two levels, while others can be used on both. It's up to the developer to find the right combination between the two sides of the development, accordingly to the application that must be created. In our live-class, most of the time has been devoted to the client side interface, to share the load between the client workstation and to lessen the load on the server.

Server side actionscript (SSA)

Server side programming is specifically used to control and manage accesses to the platform, SSA manages login procedures, determines how many simultaneous login and manages users' disconnections. The server side software does many other things, but most of them are entirely managed by the client side actionscript.

Client side actionscript (CSA)

CSA uses some methods and properties to remotely control FMS2's functionalities. There are 2 important structures at the core of the live-class functionalities:

- streaming object (StO)
- remote shared object (RSO)

StO is a one-way connection between the client's Flash player and FMS2 or between two different servers. All of the live-class interfaces access two different streaming channels that route and share over the RTMP protocol the output from multimedia peripherals (e.g. web-cam and microphone). Any stream management operation is initialized by CSA. The single clients connected to FMS2 speak to each other via RSO, i.e. a single instance's shared data on the server. The live-class is based on two different RSO types, volatile and persistent ones; both of them can activate information sharing from a disconnected state: they stop using connection bandwidth when there's nothing more to do on the client-server-clients path. When a user event occurs on the live-class interface, an event is triggered (e.g. clicking a mouse button, or moving the mouse over some areas) on FMS2, which sends a synchronization signal to one or more RSOs. To better understand this, the example below (fig. 1) shows a possible scenario where two clients, C1 and C2, connect to the same RSO1. For each connecting client, FMS2 activates the "clear" function. When, on the next step, C1 modifies a property x on RSO1, e.g. setting it to the value of 5, FMS2 sends a synchronization signal both to C1 and C2, where it says that x value has changed (there's no synchronization if x is equal to x).

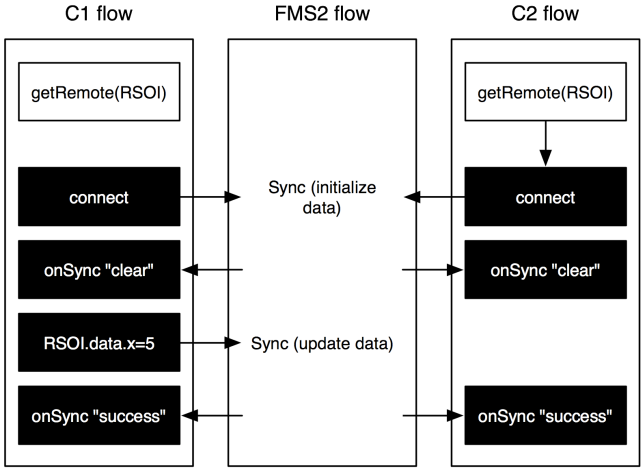

FIG.1: SHARED OBJECT FLOW

Consequently, FMS2 behaves as a variable synchronizer, and its variables are read or written by the client live-class and by the CSA. When a user triggers an event on the live-class, the linked functions activates or deactivates, accordingly to the specific RSO that has been modified, in its properties, by previous events.

C. Interface structure

Live-class web interface has been designed assembling different graphical elements on various layers. Flash allows the creation of movie clips on a frame-by-frame basis: each frame is associated to a specific function. In our live-class, the main movie-clip has the following structure:

- **frame 1**: connects the web platform with the live-class and directs different user levels towards different frames
- **frame 2**: implements the teacher interface, the one with most functionalities and that allows more intervention on RSOs and StOs.
- **frame 3**: implements the student interface
- **frame 4**: implements the tutor interface
- **frame 5**: implements the controller interface
- **frame 6**: implements the help-desk interface
- **frame 7**: implements the interface for denied access

For each type of user, a single interface is allowed. Each frame of the main movie clip contains lower level movie clips, each of whom links directly with the live-class functions.

D. Moodle + live-class integration

Since the live-class must be integrated into Moodle, a variable-passing page has been implemented, to allow communication between a php application and swf. See figure 2 for a diagram about this interaction.

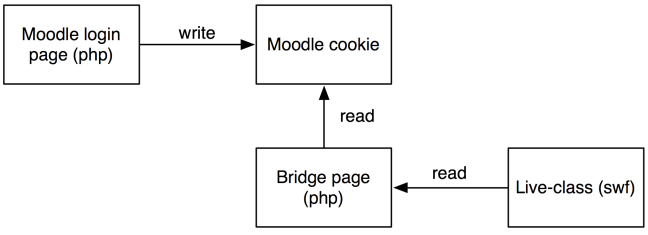

FIG. 2: MOODLE + LIVE-CLASS COMMUNICATION SCHEMA

User authentication relies entirely on Moodle, which passes to live-class's swf the user-id, if access has been granted to the live-class.

IV. Usage experience

The live-class application has been designed trying to implement in the best possible way the requirements from its users. This means that, on the basis of its users' experience (teachers and tutors), an initial requirement analysis has been performed, to define its main tools and the different user categories and privileges, followed by a study on the effort that should be made to implement them and the time this would have taken.

A. Graphical User Interface

Basing on the analysis, we identified 5 different user types, each of whom had to have its own graphical interface, with the following tools, which are necessary to perform the users' tasks.

- **Blackboard**: this is the main content-displaying area, with tools to draw and write over slides.
- **User list**: like in a chat application, this area displays the list of the users connected to the live-class. Special users (teachers, tutors and help-desk operators) are marked with graphical icons.
- **Commands**: some buttons and mini-interfaces that allow (mainly to the teacher) to manage the lesson, question students, activate hand-raising and some other specific functions.
- **Private chat**: a separate chat channel, available only to teachers, tutors and help-desk operators, to allow direct communication to coordinate the lesson, and to quickly solve configuration or network problems.
- **Public chat**: a chat channel where everybody can write, used mainly by the students to interact with the teacher when they aren't directly questioned by the teacher himself.
- **Quiz launch**: is one of the tools available to the teacher that allows him/her to evaluate students. Quizzes, which are previously set up using Moodle, are sent to students by the teacher. When the teacher launches the quiz, the live-class opens a browser pop-up window on the client computers, and answers are managed by Moodle itself.
- **Pop-up messages**: another tool which allows the teacher to send messages to all the students or to one specific student.

To improve user-friendliness of the live-class application, with the above mentioned tools we assembled many different user interfaces, one for each of the following user levels:

- **Teacher**: this is the most privileged user, being the

one who manages the lesson. Teachers have all the components and a command prompt that allows them to direct the lesson, select one student, activate audio and video and to give the student access to the blackboard. Teachers can also decide which slide to show and, using some editing tools, to launch quizzes previously defined, or even set up some quick polls to the whole class or to single students, or to send pop-up messages. See figure 3 for a relevant part of the teacher interface.

- **Tutor and help-desk**: tutors are in the same site of students and represent the trade-union between the teacher, the students and the help-desk. The latter oversees all the lessons and can interact in real time to help tutors or teachers if they have any problem. Help-desk can also do some basic troubleshooting over client computers. Both tutors and help-desk provide technical support so their GUIs are quite similar. They can take part in the lesson but they can't perform any action over the content of the lessons.
- **Students**: they are the recipients of the lesson. In SFIDE they are divided in groups, each of whom is in a different city. Their interface displays all the components and allows them to take part in the lesson. If questioned, they can talk and they are displayed in one of the two video channels, they can also interact using the blackboard if the teacher allows them to do so. They can always access the public chat channel.
- **Controller**: it's a special user, created to allow coordinators to observe the on-line lesson. The GUI is the same as students' one, but they aren't allowed any interaction with the lesson, being only a passive observer.

B. Problems and Solutions

There's no better way than usage experience to evaluate the effectiveness of an application. During the first four months of activity, the live-class and Moodle system has been evaluated and tested, collecting suggestions and requests from the different classes of users. We sorted the problems into three different categories:

- **application problems**: teachers complained about some parts of the interface being too difficult to use. Problems have been reported over single parts of the GUI, and lead to many "on the road" improvements. This process lead us to the present version 2 of our live-class.
- **architectural and structural problems**: having to deal with many users in many different sites, we had to face connectivity and bandwidth issues, partly due to having many students (up to 20) inside the same building, and over the same network link. FMS2 doesn't support multicasting, so, while our server can deliver content to large groups of users, this sometimes isn't possible due to the lack of bandwidth on the client side.
- **minimal requirements**: sometimes, even when clearly stated, client classes don't meet the minimal requirements to access the system, so we experienced slowdowns in the audio/video streams and difficulties in completing lessons. We didn't experience any problems with the classrooms that met our requirements.

V. Future directions

The many positive feedbacks we had about the SFIDE projects can allow us to foresee a future where the knowledge and the tools developed within this project can be made available to other e-learning initiatives and environments. This can lead to further improvements to the live-class, to meet different pedagogical and technological requirements. There are many possible improvements, but some main points can be taken as our roadmap for the following months.

A. More integration

At present, user profiling and live-class interface assignment is made upon a text file, which defines the appropriate GUI for any user. To improve this situation, we plan to link the live-class with Moodle, allowing the live-class to directly query the Moodle user-base and get informations about the user's profile in the particular course that integrates a live-class as one of its resources. We also could create a Moodle block displaying the real-time status of the live-class, once a course has one associated.

B. Accessibility

The current live-class allows a little personalization by the user, setting some parameters in the GUI itself. This way of operating, while good-looking, can be difficult for some users. To improve the GUI's usability, we plan to introduce keyboard shortcuts or hot-keys, allowing more keyboard interaction, to move some steps towards a more accessible (in the W3C-WAI meaning) interface.

C. Framework independence

We plan to unlink the live-class from Moodle, allowing it to be integrated into other platforms. One of the key-points of the whole ELViRA project (University of Verona's initiative for e-learning) is that it's not bound to any specific platform or communication medium, so the live-class must be considered only temporarily a Moodle-specific tool. We plan to implement a support web application, where the live-class will found one common interface to query and get informations about users. This will presumably be built as a plugin-architecture, where each platform provided by the ELViRA project will get its own plugin, to determine user's data and

role for the context where the live-class has been placed.

D. Performance and scalability

Some of the problems we noticed with the current live-class are bandwidth-related. To partially solve these problems we are evaluating two different approaches.

The most typical one is to design the system with a central server where both video streams (teacher and student) converge, and with a set of servers, possibly one for each classroom, who receive one stream from the central server and re-distribute it across multiple streams from inside the classroom itself. The main advantage is that these servers (Edge servers) share all the current live-class structure, implementing a structure distributed over the territory. The main disadvantage is the additional cost, using proprietary software and technologies.

The most innovative approach, while implemented using some well-known technologies, is to introduce multicast. While keeping the same structure we set up to manage the incoming audio and video streams (one from the teacher and one from the current student or tutor), the FMS could send both streams in "push" mode to an external streaming server, where, using IP multicast, the streams will be sent back to the clients. The main disadvantage will be creating the automatic system to set up the streams for the classes, while the main advantage will be the use of a standard way of distributing content, supported by most of the network equipment and particularly efficient, with no need to maintain a network of intermediate proxy servers.

E. Standardization

The further developments of the live-class could be even more ambitious: the platform itself could become a coordinated object container, the GUI could be re-implemented with AJAX technology, as a highly interactive web application, while graphical elements and drawings could be rendered with SVG, a vectorial format chosen by the W3C as a standard for graphics on the WWW. Audio and video streams could be managed with SMIL and standard codecs like H.264 and G728/G729, or even CELP, since most of the audio is voice. The adoption of these technologies could lead to a better compliance with W3C and ITU standards, to a reduced bandwidth usage and to improve accessibility from different client platforms.

VI. Conclusion

The SFIDE project showed us that, in some cases, an e-learning project can be developed with scarce resources, while answering to strict requirements, and that the key to its success is the sharing of the involved risks between the service provider (our University) and the users (external schools and students). Even on a low budget, the platform set up for SFIDE has been able to deliver the service with short downtimes and just two lessons over a six month period had to been rescheduled.

This project also allowed us to implement the core functionalities of a synchronous e-learning platform that we will be able to adapt and use in other projects, and that we can further improve by integrating it in the asynchronous platform we are using and by moving it towards more compliance to web standards.

VII. References

[1] Giuseppe Scollo, Giovanni M. Bianco, Riccardo Fattorini, Olga Forlani, Nicola Piccinini, Ugo Savardi, “Strategic planning and service models for the ELViRA project”, aict-sapir-elete 2005, p. 516, Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop (AICT-SAPIR-ELETE’05), 2005.

[2] Olga Forlani, Giovanni M. Bianco, Michele Albrigo, "Technology and Services: Cars and Assistance for E-Learning Roadmaps," aict-iciw, p. 12, Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW'06), 2006.