PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

ESCUELA DE INGENIERÍA

A FRAMEWORK TO FACILITATE THE DEVELOPMENT OF FACE-TO-FACE CSCL APPLICATIONS SUPPORTED BY WIRELESSLY NETWORKED MOBILE DEVICES

CLAUDIO J. ALVAREZ

Tesis para optar al grado de Master of Science in Engineering

Profesor Supervisor :

Miguel Nussbaum

Santiago de Chile, noviembre 2008

To Benedict

Acknowledgements

I wish to gratefully acknowledge the supervision of Miguel Nussbaum and Rosa Alarcón in my investigation. They both were excellent advisors, and I appreciate the enthusiasm and patience with which they guided me through the entire process.

I would like to thank the people from Eduinnova who contributed to my investigation; Raúl Santelices, for his lectures on software architecture and his willingness to help me get past the burdens of C++ programming on Pocket PCs, Sandra Suarez for providing me with top-notch equipment, and Patricio Rodríguez for his support. I would also like to thank Ximena Sibils, Daniel Cohen and Cristián Muñoz for allowing me to contribute to the new CSCL software developments for the Classmate PC platform.

I wish to express my sincere thanks to my friends from University Tomás, Andrés, Juan Pablo, Matías, Rubén, Farid, Fernanda, Carlos R., Malvoa and Claudia, Álvaro, Gonzalo C., Gonzalo V., Johann, Juan Antonio, Felipe M. and Felipe B., for their support and companionship during all these years.

I remain indebted to my parents for their advice and understanding as well as for their effort to ensure that I had an excellent education. I thank my brother Daniel for his always-good vibes, advice and willingness to help me overcome the odds, Milena and Pablo for the delicious meals they shared with me and for being cool, and my uncle Benjamín, his family and friends for all the good summers. I also wish to give my heartfelt thanks to my grandmothers for their always warm and loving welcoming.

Finally, I wish to thank John Jackson, La Reineta, my friends from the Skelletor band and my life friends Riles, Bosk and McArthy, with whom I've always had a great time.

RESUMEN

Las actividades de aprendizaje colaborativo asistidas por dispositivos móviles inalámbricos son beneficiosas para estimular el desarrollo de habilidades sociales de los estudiantes y pueden mejorar los resultados académicos. Sin embargo, el software para apoyar estas prácticas presenta desafíos a los desarrolladores, porque debe lidiar con las limitaciones de hardware de los dispositivos y al mismo tiempo asegurar un desempeño confiable y de buen rendimiento. Ante estos requisitos es posible desarrollar software a la medida de cada diseño pedagógico y optimizado para las necesidades específicas de éstos, a fin de minimizar el consumo de recursos en los dispositivos móviles. Si bien esta metodología permite implementar el software, resulta costosa, porque cuando se requiere desarrollar nuevo software, difícilmente puede reutilizarse el software implementado en una aplicación anterior. Además, el software ofrece escasa flexibilidad al momento de realizar cambios o adaptaciones para cumplir con nuevos requisitos.

En esta investigación se desarrolló un *framework* que facilita el desarrollo de aplicaciones colaborativas apoyadas por dispositivos móviles en forma económica y flexible. El *framework* promueve la modelación de diseños pedagógicos introduciendo un lenguaje que sigue los últimos desarrollos teóricos en el campo de scripts CSCL, y potencia el reuso de software y la flexibilidad para adaptar las aplicaciones a nuevos requisitos siguiendo el principio "Aplicaciones = Scripts + Componentes". Para experimentar las capacidades y el funcionamiento del *framework*, se implementó un diseño pedagógico que explota sus características y se probó en aulas tanto de nivel escolar como universitario.

Esta Tesis contó con el apoyo del proyecto CONICYT/FONDECYT 1080100.

Palabras Claves: Scripts CSCL, MCSCL, desarrollo de software, framework

ABSTRACT

Collaborative learning activities supported by wirelessly interconnected mobile devices are beneficial for stimulating the development of students' social abilities and can improve academic results. However, software for supporting these practices poses challenges to its developers, because it must be able to cope with the limitations found on mobile devices, while executing reliably and with a good performance. Given these requirements, it is possible to develop software in an ad-hoc fashion, according to the specific requirements of each pedagogical design and finely optimized for their purposes, in order to minimize resource consumption in the mobile devices. While this approach permits implementing the software, it is costly, because when new software is required, previous developments can be hardly reused. Furthermore, software offers scarce flexibility when changes or adaptations are needed to comply with new requirements.

In this investigation a framework was developed for facilitating the development of collaborative learning applications supported by wireless mobile devices in an economical and flexible manner. The framework allows modeling and implementing collaborative learning designs introducing a language that conforms to state of the art developments in the field of CSCL scripts. The language promotes software reuse and flexibility for customizing and tailoring applications to new requirements, following the "Applications = Scripts + Components" guiding principle. In order to experiment the framework's capabilities and performance in real scenarios, a pedagogical design that takes full advantage of the framework features was implemented, and tested in both school and university-level classrooms.

This Thesis was partially funded by CONICYT/FONDECYT 1080100.

Keywords: CSCL scripts, MCSCL, software development, framework

1. Introduction

1.1. Motivation

Teaching and learning methods in the traditional classroom have changed very little during the last hundred years. Despite that today's students and teachers are surrounded by an unprecedented variety of technological devices, and therefore can benefit from countless sources of information and easily communicate with the world, still to these days, when it comes to classroom education, students have to remain quiet in their seats listening to a teacher, or work individually on assigned tasks without interrupting others. Contrastingly, nowadays there is a consensus that communication, interpersonal and decision-making skills are essential for the 21st century.

We must think about ways in which technology can support students in learning not only the curricula's contents, but also social skills, such as collaboration. The latter possesses the virtue of supporting students in developing their verbal and social abilities, as well as reaching a common understanding. Consequently, creating technological tools that can effectively support collaboration between learners, and provide support for the teacher in monitoring and guiding these processes, is indeed an interesting problem.

1.2 Collaborative Learning

Collaborative learning (CL) activities can be defined as the commitment of three to five participants in a coordinated effort to reach a specific educational objective, being mutually engaged under a given set of rules and roles (Dillenbourg, 1999).

Studies suggest that when students work collaborating in groups, they obtain better academic results (Johnson & Johnson, 1999), show increased participation in group discussions, demonstrate a more sophisticated level of discourse, engage in fewer interruptions when others speak, and provide more intellectually valuable contributions to those discussions (Shachar & Sharan, 1994). When peers work in small groups they may have insight into other learner's needs, their focus, and the best way to explain (Lave & Wenger, 1991; Rogoff & Lave, 1984). The social interaction context that is fostered in CL activities generates a positive impact on learning, social behavior and motivation (Ellis et al., 1991; Miller, 2002).

Adams and Hamm (1996) and Dillenbourg (1999) have established five conditions that make for effective CL, which can be summarized as follows:

Individual responsibility: Each member must be responsible for his or her own work, role and effort to learn.

Mutual support: Each member must help in the teaching of the other members of the group.

Positive interdependence: The main goal of collaborative activities is the group goal. Therefore, collaboration is considered a success when every member of the group has interacted and accomplishes their individual goals.

Social face-to-face interactions: The decision making process must include discussions between all collaborators. Productivity is therefore influenced by the ability of the group to efficiently exchange opinions, negotiate and construct an answer together.

Formation of small groups: Communication, discussion and achievement of consensus can be only carried out in small groups, and each member of the group must be physically close to the other members.

When these conditions are fulfilled, learners see their classmates as a source of knowledge and help, rather than as a competition. They develop common values and social skills, strengthen their self-esteem and enjoy and learn more by being active participants of their learning rather than passive listeners (Zurita & Nussbaum, 2004a).

Despite the benefits that CL can bring to the classroom, small group learning is not a universally endorsed practice. At present, there is still a propensity for teachers to talk and students to listen. Galton et al. (1999) reported on a study conducted for a 20 year period, revealing that today UK teachers spend approximately 75% of their time telling students facts and ideas or giving them directions, an increase of nearly 20% over the two decades reviewed.

It seems schools are often reluctant to embrace small group learning since it is an organizational change for the teacher which appears harder and more time consuming than a traditional approach (Baine et al., 2003). This is true in part because learners do not necessarily spontaneously work together well in groups, and tasks that afford opportunities for meaningful collaboration can be hard to design and manage. The difficulty lies in developing the students' interaction skills, promoting collaborative problem solving, and providing students with emotionally and intellectually stimulating learning environments.

1.3 Computer Supported Collaborative Learning

Technology can be used to support CL activities, extending the learning experience to include communication and computing capabilities. These initiatives gave rise to computer-supported collaborative learning (CSCL) (Crook, 1994; Dillenbourg, 1999; Silverman, 1995). CSCL systems make use of technology to control and monitor the interaction between the participants, distribute information, regulate assignments, rules and roles, and, finally, promote new knowledge acquisition. However, the goal of CSCL is beyond mere technology and requires and understanding of cognition, communication and culture and its social settings to be successfully implemented.

In CSCL activities, the computer is used more as a partner than a tutor, i.e., not as a directive training means but as a way to exchange, control and build knowledge among partners (Aïmeur et al., 2001).

CSCL activities most commonly incorporate personal computers (PCs), which support the learning environment and mediate the social interactions between the group members. However, PCs are not designed for a face-to-face conversational setting (Shen et al., 2001), because the requirement that users remain behind their screens hinders face-to-face activities. In addition, there is evidence that many crucial aspects of a collaborative workplace occur when colleagues are not at their PCs (Belotti & Bly, 1996).

1.4 Mobile Computer Supported Collaborative Learning (MCSCL)

Recent work in the field of CSCL has addressed a context where the digital system is designed to support face to face CSCL, using wirelessly interconnected handheld devices (Zurita & Nussbaum, 2007), like PDAs and smartphones, seeking to overcome the deficiencies of desktop PCs in supporting CSCL activities in conversational, co-located settings. The use of wirelessly interconnected handheld devices in CSCL environments has given rise to the field known as Mobile CSCL (MCSCL) (Liu et al., 2007; Yin et al., 2007; Zurita & Nussbaum, 2003).

Studies report that collaboration among peers using handheld devices is more cooperative and friendly compared to PC-based environments (Tremaine et al., 2005). Mobile devices allow providing each student with their own computer; the mobility, flexibility and instant access provided by them means that they are ‘‘at hand,’’ which allows students to engage in highly collaborative activities anywhere, at anytime (Zurita & Nussbaum, 2003).

According to Zurita & Nussbaum (2007), the design of CL activities supported by wirelessly networked mobile devices in co-located settings, recognizes two networks; the social network, where group mates interact verbally, and the technological network that transparently supports the social network activities, by coordinating and synchronizing activity states, and mediating the activities and the social interaction of the participants.

While mobile devices help improve CSCL outcomes by eliciting highly motivating and interactive collaboration environments, the strong mobility and low invasiveness benefits they provide also bring additional complexity to the process of developing software to support collaborative activities. Mobile devices feature limited processing power, memory, battery capacity and screen size, therefore, software must succeed to overcome these limitations while providing reliable support for wireless inter-device communications, coordination and synchronization (Echeverria et al., 2006).

1.5 CSCL Scripts

Collaboration and group achievement are not necessarily accomplished by assigning students to groups and telling them to work together. One way to enhance the effectiveness of collaborative learning is to structure interactions by engaging students in well-defined scripts (Dillenbourg, 2002). A collaboration script is a set of instructions prescribing how students should form groups, how they should interact and collaborate and how they should solve the problem (O'Donnell & Dansereau, 1992).

As described by Miao et al. (2007), collaboration scripts establish an explicit contract between the teacher and the group of students regarding to their mode of collaboration. Computers are usually incorporated as mediators in the enactment of collaboration scripts, controlling access to resources (e.g. documents or tools), assigning roles to the collaborators, establishing division of labor, enforcing obedience to rules, coordinating social interactions and allocating students into groups. Computers can also track students' progress, fulfillment of script objectives, etc., and report this information to the teacher or other concerned parties.

Hernandez-Leo et al. (2007) identifies three approaches by which collaboration scripts can be implemented with technology support. In the first approach the teacher is responsible for scripting the collaboration, instructing students verbally on what they have to do, controlling their work and enforcing the script rules, while students' work is supported by technological tools such as web browsers, e-mail, spreadsheets, etc. This method does not guarantee the best results, because it generates a high workload for teachers and does not assure that students will be able to collaborate by their own means.

The second approach consists in providing automatic computer support for the scripting process, which is done by developing a new CSCL application to implement each specific script. Effective results can be obtained by this approach; nonetheless, developing specific software for each learning scenario is expensive, because new applications can hardly benefit from reusing existing functionality implemented in other ones. Software becomes rigid, offering very limited flexibility and customization options to support different learning scenarios (Haake & Pfister, 2007; Suthers, 2007).

The third approach consists in formalizing the collaboration scripts so that they can be automatically interpreted by a script engine. Computer-readable representations of collaboration scripts, known as CSCL scripts, can be achieved by defining meta-models and formal languages for representing the problem-level concepts (e.g. groups of students, rules and roles, tools, etc.) and their interrelations (Miao et al., 2007). This means that human script designers can use formal languages based on concepts they are familiar with, in order to model and specify the learning scenarios, while computers can interpret the scripts to provide transparent support (mediation) for the learning activities (Kobbe et al., 2007).

Formalizing CSCL scripts has significant advantages compared to the previously mentioned approaches. In the first place, script-based software is easier to

compose, customize and adapt to new instructional designs, because scripts are described in terms of domain-specific languages that facilitate developing tools based on notations that are familiar to instructional designers, education researchers, and teachers (Hernandez-Leo et al., 2007; Miao et al., 2007). Secondly, establishing a scripting language ensures that all scripts that conform to the language adhere to a common conceptual framework. This facilitates sharing and interchanging instructional designs, as well as software reuse; therefore, costs and efforts necessary to experiment with new pedagogical approaches can be lessened considerately. Finally, CSCL scripts are resistant to technological change, because the use of script engines allows them to remain independent of the underlying hardware and operating system platforms. In other words, it is not necessary to modify the scripts to support different operating systems, however, a script engine must be provided for each operating system where the scripts are required to operate. Fortunately, with current technology it is possible to create platform independent script engines (Blom et al. 2008).

1.5.1 CSCL Scripts in MCSCL Environments

CSCL Scripts are a convenient means towards creating computer-mediated CL designs in a flexible and economical way. However, up to now there has been a lack of support for CSCL scripts in co-located settings supported by wirelessly networked handheld devices. Script engines used to implement CSCL scripts are based on E-Learning standards, such as IMS LD (2003). These standards are focused on supporting both distance and co-located learning activities by means of Web-centric technologies, requiring full fledged server backend systems, and are therefore not adequate for settings where Internet connectivity is not available, or it is not possible to incorporate powerful servers in school environments.

1.6 Hypothesis

It is possible to develop an application framework to facilitate the implementation of CSCL scripts in a face-to-face classroom environment supported by wirelessly networked mobile devices.

1.7 Objectives

The main objective of this investigation is to develop an application framework that facilitates the implementation of CSCL scripts in a face-to-face classroom environment supported by wirelessly networked mobile devices. For this purpose, the framework must offer software developers an architecture that can withstand the requirements of such applications, and provide a core of common functionality required by them. Accordingly, it is mandatory to resolve the requirements of CSCL scripts that must be taken into consideration in the framework specification, determine the most adequate architectural design approach for the framework, develop the framework ensuring that it is suitable to implement the intended scripts, and finally, test the framework by means of implementing a CSCL script that can leverage its features.

1.8 Methodology

The process by which the objectives of this investigation were accomplished followed five successive steps: (1) Literature review in the state of the art of CSCL script conceptual frameworks and scripting languages. (2) Determination of a meta-model for modeling CSCL scripts in co-located settings. (3) Definition of a scripting language for representing CSCL scripts. (4) Design and implementation of the application framework. (5) Implementation and testing of a CSCL script by means of the framework. The following pages present an in-depth account on how the five steps were accomplished.

1.8.1 Literature Review

In order to determine the requirements of the application framework subject of this investigation, it was necessary to review the state of the art in the fields of CSCL script conceptual frameworks and scripting languages.

1.8.1.1 CSCL Scripts Formalization

A broad description of CSCL scripts was found in Dillenbourg (2002), who in a pioneering effort described the components of CSCL scripts and their interrelations. According to Dillenbourg (2002), syntactically, a script is a sequence of phases and each phase can be described by the following attributes:

Task definition: Task assignments are defined as a triplet composed by input, activity and output elements. The input includes the pre-requisites for realization of the activities that conform the task (e.g. a previous result, a document, etc.). Activities are prescribed work units that present learners with goals or sub-goals framed in the task or script in which they are defined. The output refers to the results produced as a result of the collaborative effort in the realization of an activity (e.g. products achieved as a result of goal fulfillment).

Group definition: Refers to the criteria and mechanisms through which groups of students are formed. Two group formation criteria are described; the external criteria, which distributes students on the basis of some students' characteristics that pre-exist the CSCL activity, and the internal criteria, which distributes students according to their behavior or products that have been collected in a previous phase of the script.

Distribution: Establishes how input and/or activities are distributed among learners, and whether this is done in an inter-group or an intra-group manner.

Mode of interaction: Determines the forms of communication, coordination and synchronization available to learners in the successive activities and tasks of the scripts (e.g. computer-mediated distance interaction and face-to-face interaction).

Timing: Refers to the duration or output delivery time of activities and tasks that conform the scripts.

The grammatical combination of these elements may however produce any kind of pedagogical method, even those that have nothing to do with collaborative learning. Seeking to establish criteria by which scripts can express collaborative learning methodologies, Dillenbourg (2002) identifies four semantic aspects that reflect on the pedagogical meaning of the scripts:

Design rationale: Alludes on the hypothesis upon which a CSCL script is designed. In the most ample sense, CSCL script hypotheses are about learning from social interactions; they explain how scripts foster some social interactions and inhibit others, and the way in which the expected interactions are expected to produce learning effects.

Coercion degree: Establishes the degree of freedom that the learners have in following the script. A high degree of coercion may harm the natural social interactions arise in collaborative environments, whereas a low degree of coercion may be insufficient to significantly influence the collaborative process.

Appropriation: Determines whether a script is simple of difficult for students and tutors to appropriate. The first level of appropriation is adoption, which says that students and tutors have to understand the script without additional difficulty or overload. The second level of appropriation is internalization, which refers to the ability of students and tutors to individually and internally replay the cognitive processes in which they participated during the script. In other words, determines whether the script itself is easy to learn or not.

Generalisability: Refers to the possibility of applying a script to different knowledge domains.

In accordance with the definitions given by Dillenbourg (2002), in the design and implementation of CSCL scripts there are semantic aspects that concern the pedagogical design, which are expressed through syntactic components. Therefore, the framework subject of this investigation must offer support for expressing pedagogical designs through well-defined syntactic components of the MCSCL domain that can be embedded in the software.

Kobbe et. al (2007) extended the formulation by Dillenbourg (2002) and proposed a generic framework for specifying collaboration scripts. The framework takes into account both syntactic and semantic aspects of scripts establishing different abstraction levels. Situated at the highest abstraction level are the script schemata, which can be understood as patterns that represent pedagogical designs in a generalisable way, such as the Jigzaw method (Aronson & Patnoe, 1997). Script schemata can be instanced through components (participants, activities, roles, resources and groups) and mechanisms (task distribution, group formation and sequencing), both which are found in a lower abstraction level.

Considering the components and mechanisms defined in the framework presented by Kobbe et. al (2007) a relation can be established with the Expanded Mediation of Activity Theory (AT) developed by Engeström (1988). Upon the basis of recent works on the application of AT to mobile learning (Uden, 2007), CSCL (Caeiro-Rodríguez et al., 2007) y MCSCL (Zurita & Nussbaum, 2007), we complemented the

information provided by Dillenbourg (2002) and Kobbe et. al (2007), seeking to attain a broader perspective for determining the requirements and design considerations for enabling collaboration scripts in MCSCL environments. More details on these developments can be found in section 2.3.

1.8.1.2 CSCL Scripts Implementation

Up to recently, concrete attempts to implement CSCL scripts by means of a formal language have been based on the IMS LD (2003) standard (Hernandez-Leo et al., 2007; Miao et al., 2007). IMS LD is a process modeling language that is used to model units of learning (UOLs), being sufficiently expressive to describe a wide range of pedagogical approaches.

It was determined that IMS LD presents three major difficulties for developing CSCL scripts in co-located settings supported by wireless mobile technology. In the first place, the IMS LD meta-model is complex as it aims at generality in providing expressiveness for different kinds of instructional designs, therefore, the compliant runtime environments are difficult to implement and require enterprise-class servers to run the UOLs (Griffiths et al., 2005; Martens & Vogten, 2005). Secondly, IMS LD runtime environments are based on a Web-based client-server architecture, which does not guarantee appropriate network performance in an operational environment comprising a large number of mobile devices interconnected in ad-hoc wireless networks (Echeverria 2006). Finally, IMS LD despite offering powerful pedagogical expressiveness fails to provide adequate support for CSCL scripts (Hernandez-Leo et al., 2007; Miao et al., 2007; Caeiro-Rodriguez et al., 2007). Miao et al. (2007) identifies five weaknesses in the IMS LD language, which make it unsuitable for expressing this kind of learning designs:

Modeling groups: The language offers no syntactic element that can explicitly represent groups in the learning design. Groups are constructed by simultaneously assigning a role to a series of individuals, therefore, groups must be modeled implicitly. Furthermore, the number of people in the groups must be known beforehand at design time, and the language does not allow modifying the group composition at runtime.

Modeling artifacts: In learning processes, teachers and learners usually generate artifacts such as a document, a vote, an answer, an argument, etc. As with groups, IMS LD does not offer explicit support for modeling artifacts.

Modeling dynamic features: With IMS LD is difficult to implement non-deterministic script behavior, including support for 'imperfect' social structures, which is necessary in order to guarantee flexibility of CSCL scripts (Dillenbourg & Hong, 2008).

Modeling complicated control flow: With IMS LD is difficult to specify non-linear flow of activities (e.g. loops).

Modeling various forms of social interaction: With IMS LD it cannot be clearly modeled whether and how people collaborate, because it lacks the expressiveness

to model social structures and their interactions in different modes (i.e. individual work, group work, classroom work).

In order to overcome the deficiencies found, Miao et al. (2007) proposed a domain-specific language to formalize CSCL scripts. The language overcomes the deficiencies by explicitly introducing components of CSCL scripts that IMS LD fails to describe appropriately. These components introduced by Miao in his language are the following:

Groups: The language explicitly allows modeling groups and configuring them by introducing attributes such as max-size, min-size, form-policy, disband-policy, dynamic/static, among others. Groups can be assigned into roles and have sub-groups to form a hierarchically structured organization.

Artifacts: In Miao et al. (2007)'s language, artifacts are treated as files that can conform to known MIME-types or user-defined types. Aggregated artifacts allow appending collective information to the same file, therefore, it is possible to implement shared artifacts.

Actions and expressions: Actions are included in the language to address the limitations of IMS LD in defining dynamic script features. They are used to define arbitrary procedures that can be executed by the runtime system, allowing to query and manipulate the state of activities, artifacts, roles, persons, transitions, environments and their interrelations.

Transitions and routing activities: The language introduces transitions and routing constructs recommended by the Workflow Management Coalition (WFMC Web site), which allow modeling complex control flow in CSCL scripts.

Activity-centered methods to assign roles: The language allows establishing interaction rules among different roles, and therefore explicitly describing how different persons or groups can collaborate.

Due to the both technological and conceptual drawbacks of IMS LD for developing CSCL scripts in co-located environments supported by wirelessly-networked mobile devices, in this investigation it was decided that the most convenient approach to specify a language for implementing CSCL scripts is by means of a formalization that takes Miao et al. (2007)'s concepts and ideas into consideration, however, making adequate technical simplifications to his model so that CSCL scripts can operated in mobile devices.

1.8.2 A meta-model for specifying CSCL scripts

In order to define a language to formalize CSCL scripts, it is necessary to define its syntax and semantics (Dillenbourg 2002). In this investigation, this was done by specifying a meta-model defining the building blocks of CSCL scripts and the interrelations between them (Miao et al., 2007). Further details concerning the meta-model developed in this Thesis can be found in section 2.4.

1.8.3 A formal language for representing CSCL scripts

For the purpose of representing CSCL scripts in a machine-interpretable way, a formal language that can be understood and applied by script developers is required. In this investigation, the pertaining language was specified in terms of the meta-model for CSCL scripts described in section 2.4.

The language developed in this investigation, called ELFML (ELF Markup Language), is defined by an XML schema (see Appendix A) and therefore has an XML-based syntax. A detailed description of the elements that compose the ELFML language can be found in section 2.5.

1.8.4 Design and Implementation of the application framework

Having established a meta-model that defines the building blocks for modeling CSCL scripts and a scripting language to describe them, the following step in this investigation consisted in developing an application framework for implementing the scripts.

In order to facilitate development of CSCL scripts, it was determined that the framework had to fulfill four requirements. In the first place, provide support for the ELFML scripting language and therefore, include an interpreter capable of parsing the scripts and controlling their execution. Secondly, support the variation points (see section 2.4) defined by the meta-model, and consequently, provide APIs necessary for implementing the different kinds of components allowed by them. Thirdly, provide a middleware that can handle network communications appropriately, given the known limitations of mobile devices and performance of wireless networks in settings comprising large number of interconnected mobile nodes (Echeverria et al., 2006). Finally, the framework also had to provide basic functionality required for establishing sessions, allocating learners into groups and synchronizing overall script execution among all the participating network nodes.

A description of the framework architecture, APIs and functioning can be found in section 2.6.

1.8.5 Implementation and testing of a CSCL script

In order to test the application framework developed in this investigation by means of applying it to a concrete problem, the CollPad CSCL script described in section 2.7.1 was implemented in the final stage of the methodology followed in this Thesis.

The script aims at facilitating collaborative problem resolution in the classroom environment. It randomly organizes the students into groups of three, and scaffolds their work and interactions through a sequence of well-defined phases involving individual work, group discussion and classroom discussion, leading to a final solution to the original problem that represents a classroom-level consensus.

The script presented the following fundamental requirements for its implementation:

Support for organizing students into randomly assigned groups of three,

creation, transmission and distribution of artifacts (problem and solution representations) over the wireless network,

and support for handwriting on the touch screen of the mobile device to generate artifacts.

All three requirements listed above could be satisfied by the framework developed in this investigation. An account on the process by which the script was implemented and tested can be found in sections 2.7.2 and 2.7.3, respectively.

1.9 Results

An application framework to facilitate the development of co-located CSCL scripts supported by wirelessly networked handheld devices was the result of this Thesis. The framework was successfully implemented, fulfilling the requirements mentioned in section 1.8.4.

Due to the high reuse of framework elements (e.g. network layer, session management and group management layers) and the possibility to design with the pre-established meta-model, the effort required for implementing CSCL scripts using the technology developed in this investigation is less than developing the script by means of an ad-hoc application in the traditional manner, considering that the ad-hoc application would need to implement much of the functionality already provided by the framework besides its own logic, user interface, etc.

In the implementation of implementation of the CollPad script, the framework facilitated the development work at both design and programming times. During the design time, the meta-model could be naturally applied to the problem; therefore, determining the components required to implement the script's specific requirements, and designing them, was a straightforward process. On the other hand, at programming time the script components benefited from (reused) all the core features of the framework, like network protocols, group and session management logic, and script synchronization. In addition, the framework APIs facilitated developing new components (e.g. tools, artifacts, actions) when the framework could not provide specific features required by the script. Finally, the ELFML language allowed the developer to integrate the different components of the application with great flexibility.

1.10 Conclusions

Scripted collaboration is currently a broadly researched topic in the CSCL community. Nevertheless, up to now little efforts have been made to promote use of CSCL scripts in co-located collaboration environments supported by wireless mobile technology. In this Thesis, a technology was developed with the aim of facilitating the design and implementation of CSCL scripts in environments of such characteristics, in a flexible and economical way.

The formalization of CSCL scripts developed in this investigation through the meta-model and the scripting language proposed, brings various benefits. In the first place, the formalization achieved raises the abstraction level upon which the software is implemented. This means that developers require spending less time in software design tasks, in comparison to the approach of developing a specific application for each specific script, because the software can be designed rapidly leveraging the problem domain concepts introduced beforehand by the meta-model. Secondly, the formalization establishes a common conceptual framework to specify CSCL scripts for co-located settings, therefore it facilitates sharing and interchange of instructional designs among researchers and practitioners. Lastly, if well the meta-model and the ELFML language were specified defining hotspots as a strategy to maintain software small and so overcome handheld device limitations, this does not restrict creation of scripts in the ELFML language to this kind of devices. The same approach is valid for developing CSCL scripts for more capable devices, such as Tablet PCs and Ultra-Mobile PCs (UMPCs).

The framework developed in this investigation includes a middleware that allows developers focusing their efforts in the particular requirements of the scripts, relieving them from implementing software to support basic services and functionality, such as network protocols, group and session management logic, synchronization mechanisms, etc. In addition, the highly modular architecture that is fostered by means of the "Applications = Scripts + Components" guiding principle (Nierstrasz & Achermann, 2000) leads to easily parallelizable software development. For teams comprising multiple developers, it is possible to simultaneously work on different script plug-ins either individually or cooperatively, thus increasing overall productivity.

Although the layered software architecture that is necessary to execute ELFML scripts on mobile devices is sophisticated, as it blends a network middleware with a scripting language interpreter, current PDAs and smartphones are capable of supporting it, executing the applications with no user-noticeable sacrifice in performance.

1.11 Future Work

The framework developed in this Thesis is based on the Edunova Network middleware that if well provides good network performance an reliability, limits the possibility to execute the framework (and the CSCL scripts) in other mobile platforms, such as UMPCs or Tablet PCs, due to its strong dependence on the Windows Mobile operating system, which is found exclusively in PDA and smartphone devices. Aiming at software portability, a significant step for future development of this technology would be releasing a version of the framework that can run independent of the subjacent operating system. This could be accomplished by providing an implementation that depends on the Microsoft .NET Framework, which is however only fully supported by Windows-based platforms, or the Sun Java Platform, which is supported by most current operating systems.

In order to easier the development of CSCL scripts using the technology proposed in this Thesis, a necessary step would be developing a set of ELFML-compliant visual

authoring tools to compose both the flow of activities and the environments that conform the CSCL scripts. However, if this might well help developers produce ELFML automatically, script plug-ins will remain as a significant bottleneck in the attempt to implement CSCL scripts effortlessly, because they need to be developed by means of traditional object-oriented programming. Moreover, since each script is the reflection of a specific pedagogical design, it is not trivial to create plug-ins (e.g. Action classes) that can be easily reused in different scripts (Bote-Lorenzo et al. 2004). One way of tackling these difficulties would be to provide plug-in libraries and reusable script templates to support generalisability of pedagogical designs. This could be done by developing support for common activities found in collaborative learning scripts in the form of collaborative learning flow patterns (CLFPs) (Bote-Lorenzo et al., 2004; Hernandez-Leo et al., 2007), or the abstract script schemata proposed by Dillenbourg & Jermann (2006). In this way, script developers can take advantage of pre-defined pedagogical designs and tailor them to fulfill the needs of the desired learning scenarios.

2. A framework to facilitate the development of face-to-face CSCL applications supported by wirelessly networked mobile devices

2.1 Introduction

Collaborative learning describes a *situation* in which particular forms of interaction among people are expected to elicit learning. However, in these situations there is no guarantee that the expected interactions will actually occur (Dillenbourg, 1999). One way to enhance the effectiveness of collaborative learning is to structure interactions by engaging students in well-defined scripts. A collaboration script is a set of instructions prescribing how students should form groups, how they should interact and collaborate and how they should solve the problem (Dillenbourg, 2002).

Collaboration scripts have become fairly popular within educational science, especially in the domain of computer-supported collaborative learning (CSCL) (Kobbe et al., 2007). Software for supporting scripted collaborative learning is designed with the aim of scaffolding social interactions (Dillenbourg, 2002), by controlling the user-interface and activities available to the learners according to the sequences of activities defined in the scripts (Haake and Pfister, 2007). Being embedded in the learning environment, scripts can structure social interactions and support the learners with the activities in which they engage (Kobbe et al., 2007; Suthers, 2007).

Scripted CSCL environments are typically designed to support collaboration by incorporating personal computers (PCs) in distance or face-to-face settings (Dillenbourg, 2002; Haake and Pfister, 2007; Kobbe et al., 2007). The use of PCs requires users to remain seated behind a screen and adapt their interactions to the single-user paradigm most PCs are based on (Zurita and Nussbaum, 2004a). Aiming to overcome these deficiencies, studies have shown that wirelessly networked handheld devices, such as PDAs and smartphones, can support face-to-face interaction to further enhance outcomes of CSCL, giving rise to MCSCL (Mobile CSCL) (Liu et al., 2007; Yin et al., 2007; Zurita and Nussbaum, 2004b).

MCSCL activities take full advantage of mobile devices and wireless connectivity to support small group collaborative learning. The use of wireless ad-hoc networks allows performing the activities anywhere and with a large number of people. However, developing technological tools to support these learning scenarios is challenging, because software must be able to cope effectively with mobile device limitations (e.g. limited processing power, memory, small screensize, etc.), while ensuring that reliability and a trustworthy performance are always met (Echeverria

et al., 2006). Although developing finely-tuned tools specialized for the needs of each particular collaborative activity is possible, this approach is costly, because of the limited flexibility that it offers for responding to changing requirements, and the low software reuse that may be achieved when implementing new applications.

CSCL scripts are computer-readable formalizations of collaboration scripts (Kobbe et al., 2007). These are specified in terms of domain-specific languages, which allow describing instructional designs by representing problem-level concepts and interrelations with a formal notation that practitioners, such as education researchers and technical experts can understand and apply to real problems (Hernandez-Leo et al., 2006). CSCL scripts are run by script engines that provide the necessary functionality and services required by the scripts. This means that scripts themselves do not require addressing lower-level concerns such as querying databases and managing network connections. As a result, the scripting approach allows creating flexible software that can be rapidly developed, and easily customized and tailored to fit new learning scenarios.

CSCL scripts are a convenient means towards developing software tools for supporting collaborative learning, however, there is still a lack of support for implementing scripts in MCSCL environments. Current implementations of CSCL scripts are based on complex meta-models (Dan and XinMeng, 2007), hence, the underlying technology that permits the execution of CSCL scripts is mostly based script engines that rely on enterprise-class application servers and databases (Griffiths et al., 2005; Haake and Pfister, 2007). Evidently, this technology is inadequate for classrooms lacking Internet connectivity or where servers cannot be afforded.

In this article, we present an application framework based on interpreted CSCL scripts aimed at facilitating the development of MCSCL applications in a flexible and economical manner. The framework includes a middleware that allows executing the scripts in mobile devices interconnected by WiFi (802.11) networks in peer-to-peer or infrastructure modes, a script interpreter that controls script execution, and APIs for extending the framework functionality through plug-in development. The scripting approach follows a component composition strategy that conforms to the "Applications = Scripts + Components" guiding principle (Nierstrasz and Achermann, 2000), leading to flexible software that can quickly respond to evolving requirements. To illustrate our approach, we present the CollPad CSCL script (Nussbaum et al., 2008a) as a case study, giving an account of the implementation process, making notice of the benefits and drawbacks encountered.

2.2 CSCL Scripts in Mobile Environments

Collaboration scripts have become a major topic in the research community on CSCL (Kollar et al., 2005). One main reason for this is that the script concept plays a specific role in all three disciplines that converge in CSCL: cognitive psychology, computer science, and education. It is therefore an anchor point among multidisciplinary and interdisciplinary discourse (Fischer et al., 2007).

Computer science tries to develop formal languages and devices that support designers and practitioners in easily setting up collaboration scripts for computer-mediated learning (Fischer et al., 2007). With this aim, Educational Modeling Languages (EMLs) are intended to support the modeling of educational units in general, including collaborative learning approaches.

The most relevant attempts to formalize CSCL scripts are based on the IMS LD standard (IMS LD, 2003). IMS LD is a process modeling language, which aims at describing a wide range of learning designs associating educational content with their instructional strategy in a consistent and machine-interpretable way (Dan and XinMeng, 2007). A teaching-learning process can be codified into an XML file with references to the learning environments needed to perform activities, and executed by any runtime environment compliant with the IMS LD information model and schemas. Nevertheless, current IMS LD support for CSCL is a topic of major research interest marked by controversy (Hernandez-Leo et al., 2007). Recent investigations from different researchers agree that IMS LD lacks the expressiveness required to describe the domain of CSCL scripts comprehensively, and have proposed solutions to the problems encountered (Caeiro et al. 2003; Hernandez-Leo et al., 2007; Jurado et al., 2006; Miao et al., 2007). However, the IMS LD standard specification has remained unchanged since its 1.0 release back in 2003.

MCSCL environments, as described by Cortez et al. (2004), Liu et al. (2007) and Zurita and Nussbaum (2004b) introduce wirelessly interconnected mobile devices to support face-to-face collaborative work in the classroom. Students organized in small groups are assigned a handheld device connected to a wireless LAN, which is usually WiFi in ad-hoc (peer-to-peer) or infrastructure mode. The teacher also has a handheld device, from which he/she can monitor the students's work. The considerable amount of devices that concur in the wireless network causes that network performance and reliability are notably inferior to wired LAN networks that are used in CSCL environments equipped with PCs. This poses a problem, because network trustworthiness and performance are of great importance for the adequate implementation of collaborative activities (Echeverria et al., 2006).

The mobility provided by handheld devices is beneficial for implementing face-to-face collaborative activities (Zurita et al., 2003), however, mobile devices present constrained hardware if compared with laptops or desktop PCs (Vavoula and Karagiannidis, 2005). Considering the limitations of the mobile devices, as well as the importance of assuring a reliable network performance during the collaborative activities, the implementation of CSCL scripts in an MCSCL environment requires developing a script engine that includes a middleware able to cope effectively with the given device limitations and provide trustworthy network performance. In addition, it is desirable that the devices can autonomously run the script engine, without requiring a full-fledged server, because server equipment is invasive, expensive and difficult to operate. It is also important to consider that classrooms do not necessarily incorporate Internet connectivity; therefore it is not appropriate to depend on remote servers to execute the activities.

The IMS LD language can be used to describe a wide variety of learning activities. Nevertheless, the powerful expressiveness of the language is possible in virtue of

the complex underlying specification that defines it; hence, script engines are sophisticated and difficult to implement (Dan et al., 2006; Griffiths et al., 2005; Martens and Vogten, 2005). Due to the complexity of the script engines required by the IMS LD standard, and the fact that the expressiveness of IMS LD to describe CSCL scripts has been criticized, in order to support CSCL scripts in MCSCL environments it becomes necessary to develop a simpler scripting language without sacrificing the benefits of flexibility, reuse, customization and tailorability that CSCL scripts bring forth.

2.3 Specifying a Language for Mobile CSCL Scripts

A CSCL scripting language is in practical terms a language that allows specifying the way in which computers mediate, monitor and control collaborative activities (Kobbe et al., 2007). When CSCL scripts are operationalized, technology can scaffold interactions between learners in accordance to the prescribed instructions of the script. CSCL scripting languages are useful to describe CSCL scripts when they can be understood and applied by developers and provide with the sufficient expressiveness to describe the intended machine behavior for the collaborative learning scenarios (Caeiro-Rodriguez et al., 2007; Haake and Pfister, 2007; Miao et al., 2007).

A language for specifying CSCL scripts can be understood and applied more easily if it is based in a meta-model that faithfully represents the problem domain (France and Rumpe, 2007). This means that the expressiveness of a CSCL language will be limited by the composition of the meta-model upon which it is defined. In our quest for specifying a language to formally describe machine behavior in scripts for MCSCL environments, we developed a meta-model that establishes the conceptual basis for our scripting language. The meta-model is based on the assumption that face-to-face collaborative learning activities, such as MCSCL activities, are characterized by having a strong social interactions component, i.e. a "social network", as described by Zurita and Nussbaum (2007). Therefore, it is strictly concerned with the role that technology assumes for supporting the social component of these activities.

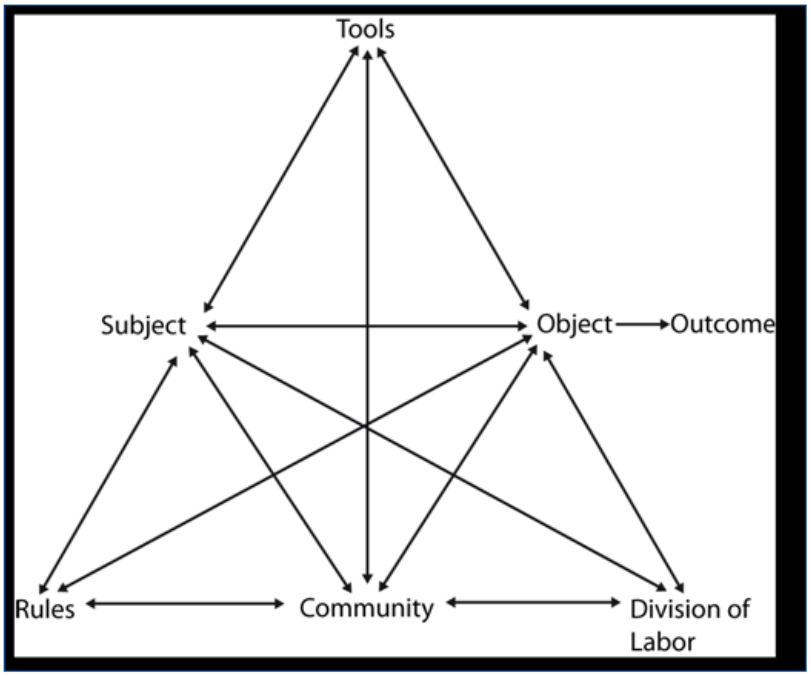
2.4 A Meta-Model for Mobile CSCL Scripts

Recent investigations have relied on Activity Theory for modeling collaborative learning (Caeiro-Rodriguez et al., 2007), mobile learning (Uden, 2007) and MCSCL pedagogical designs (Zurita and Nussbaum, 2007). The Activity Theory is a meta-theory for analyzing human practices (activities) and their constituent components. Considering that any educational unit can be conceived as a set of activities, we can use Activity Theory for describing CSCL scripts in terms of the activities that compose them, and in this way specify the requirements of the meta-model we seek to develop.

The Expanded Mediational Model (Engeström, 1987) provided by Activity Theory (Figure 2-1) holds that every human activity involves a subject (e.g. person), represented by a role that acts in an object (e.g. collaborative activity) in order to achieve a certain outcome. The relationship between the subject, his/her role and

the object is mediated by an environment that provides tools and resources the subject can use, and a community that defines the social context in which the subject acts. The rules impose regulations that affect the way in which subjects interact in the community, including their relationship with the environment. The division of labor is related with the decomposition of the activity goals into sub-goals and the distribution of responsibilities among the subjects (Caeiro-Rodriguez et al., 2007).

Figure 2-1 - Engeström's Expanded AT Model



CSCL scripts follow a temporal structure (Dillenbourg and Hong, 2008). For instance, a script may be composed by a series of sequentially ordered activities, each of them producing a result required by the next one. Scripts may also sequence activities following more complex patterns by use of conditional branching and loops. The way in which script activities are sequenced is referred as to the script flow. Besides including the elements necessary for modeling CSCL activities according to the AT components, our meta-model must also consider the notion of script flow in order to allow the temporal ordering of activities.

In the context of face-to-face collaborative activities supported by wirelessly networked handhelds, we can analyze the way in which technology mediates in the components of the AT Expanded Mediational Model:

Rules: Technology is used for enforcing the rules of the collaborative activity to the subjects that interact within the community. This is related to the fact that CSCL scripts impose a coercion degree to social interaction, in order to foster interactions that lead to successful collaboration (Dillenbourg, 2002; Dillenbourg and Hong, 2008).

Subject: In collaboration scripts, students participate in the social structure as collaborators and the teacher acts as a tutor and a mediator (Dillenbourg, 2002; Dillenbourg and Tchounikine, 2007; Zurita et al., 2005). Every subject is represented by at least one role, e.g., the teacher and student roles in the broadest sense. The notion of "role" is a synthetic way to differentiate the contributions expected from the members of the community (Dillenbourg and Hong, 2008). Technology is an active mediator in role control and assignment, because it coordinates role distribution and enforces rules according to the behaviors that are allowed to the different roles.

Community: Four basic elements, namely students, groups, the whole class and roles define the social structure of any collaboration script (Dillenbourg and Hong, 2008). Technology can configure the social structure in which these elements relate with each other, given the particular requirements of the script activities (Zurita and Nussbaum, 2004a). For example, it is common that in MCSCL activities subjects are organized in small groups according to a group allocation policy managed by the software (Zurita et al., 2005).

Tools: In CSCL scripts, subjects as well as the community in which they are engaged consume tools and resources (e.g. educational content, artifacts) that technology provides them (Dillenbourg, 2002; Dillenbourg and Hong, 2008; Hernandez-Leo et al., 2007). Technology coordinates the way in which the community can access tools and resources; for instance, by establishing floor control and flow of artifacts mechanisms.

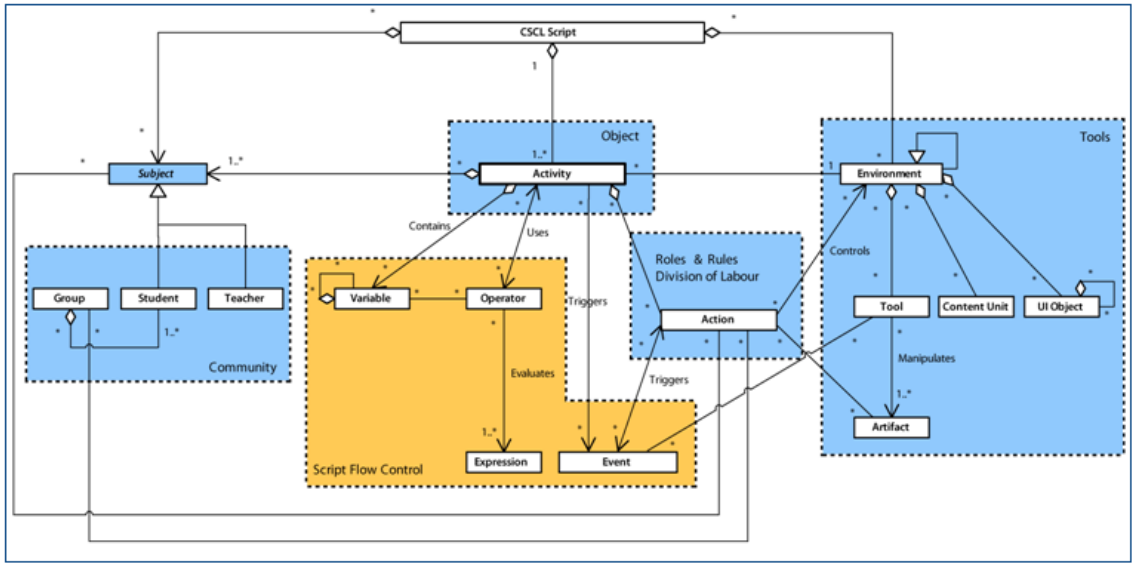
Division of labor: It is common that technology controls and monitors assignment of tasks among the subjects that integrate the community with different roles (Dillenbourg, 2002; Dillenbourg and Hong, 2008).

Object: Technology coordinates all of the above transparently, in order to support the realization of collaborative activities achieving the desired outcomes (Hernandez-Leo et al., 2007; Suthers, 2007).

Figure 2-1 presents our meta-model towards defining a language for MCSCL scripts. Our approach is inspired by the CSCL scripting language that Miao et al. (2007) proposed as a response to the shortcomings found in IMS LD for describing CSCL scripts. Like in that approach, we explicitly introduce artifacts, groups and activity routing (script flow) mechanisms in the meta-model.

It can be observed that all components of the AT Expanded Mediation Model are represented by interrelated meta-classes. However, in this initial attempt to establish a meta-model for MCSCL scripts, the Rules & Roles and Division of Labor components are implicit, as they must be implemented in each particular script as instances of the Action meta-class. Like in the model proposed by Miao et al. (2007), we introduce the Action meta-class as a generic, granular and flexible alternative for introducing developer-defined script behavior. We made this design decision for the sake of simplicity, in order to derive a minimalistic script engine; however, in the future we could introduce explicit meta-classes for modeling rules, roles and division of labor elements.

Figure 2-2 – Our meta-model towards a language for MCSCL scripts



2.5 Scripting Language Specification

Considering the meta-model as a conceptual basis, we defined the syntax of our scripting language, which we have called ELFML (ELF Markup Language), based on XML. We preferred XML for practical reasons; in the first place, XML is well established as a meta-language that facilitates interoperability between different software platforms. In fact, XML parsers and validators are available in most of current operating systems. Secondly, it has become a common practice to use XML for keeping object graphs serialized in persistent storage (van Engelen, 2004). This characteristic can facilitate future development of script authoring tools, such as visual editors, which is desirable given the efficiency and ease of use that this tools allow, in comparison to writing script code using text-based tools (Hernandez-Leo et al., 2006).

ELFML scripts instance the meta-model shown in Figure 2-2 through the XML tags on Table 2-1. Most of the meta-classes defined in the meta-model are represented in ELFML with an XML tag. However, there are meta-classes that are not explicitly represented in the language, such as the Artifact or Event meta-classes, which are intended to be instanced and manipulated by means of an object-oriented language such as C++ or Java. Table 2-1 shows a description of each tag in the language and its relationship with other tags. For example, the <activity> tag instances a class from the Activity meta-class, which may contain Actions, Operators and Variables as it is defined by the meta-model.

To control the script flow, the language provides transition operators (e.g. Goto, Forward and Back) and logic operators (Switch and If) that are instanced with <operator>. Actions can control Environment objects, such as Tools and UI Controls, and the latter can notify actions through events, in an analogous way to

the Controller-View relationship defined by the MVC architectural pattern (Gamma et al., 1994).

Table 2-1 - ELFML Tag Definition. Each XML tag that constitutes the language is presented with the attributes it can admit, and the children tags it may contain.

Tag	Description	Attributes	Children Tags
<script>	The root element of an ELFML script.	id, strictELF	<meta>,<include>,<teacher>, <student>,<activity>, <environment>
<meta>	Adds metadata to the script, in the form of {key, value} pairs.	key, value	
<include>	Includes plug-ins and external scripts into the declaring script.		<plugin>, <file>
<file>	Specifies a file (e.g. external script) that can be included within the	source	
<plugin>	Includes a plug-in that can extend the script engine functionality.	source, name	
<activity>	Declares an activity. If the entrypoint attribute is set, the script execution starts from that activity.	id, name, environment, entrypoint	<action>, <actionseq>, <operator>, <variable>, <groupConfiguration>,
<action>	Declares and configures an action to be executed within an activity.	id, name, pluginId, class,	<param>
<actionseq>	Declares an action sequence to be executed within an activity.	id, name, activationMode, automatic	<action>
<param>	A configuration parameter in {key, value} format.	key, value	
<environment>	Specifies an environment for use by an activity.	id, name, base	<contentUnit>, <uiControl>, <tool>
<contentUnit>	Specifies a content unit for use within an environment.	id, name, source, layer, renderer	<rect>, <param>
<tool>	Declares and configures a tool for use within an environment.	id, name, pluginId, class,	<rect>, <param>
<uiControl>	Specifies a UI object for use within an environment.	id, name, pluginId, class,	<rect>, <param>
<eventListener>	Declares and configures an event listener which can delegate event processing to an action.	id, eventName, activationMode	<param>
<groupConfiguration>	Can be only declared within activities enclosed by the <teacher> tag. It sets/modifies the student group composition.	id, name, pluginId, class, policy	<param>
<variable>	Declares a variable and optionally sets its initial value.	name, value, type	
<operator>	Declares and configures an operator. The <param> child elements can specify the operands and the variables	id, name, pluginId, class	<variable>
<teacher>	May contain one or more <activity> child elements intended to be executed		
<student>	May contain one or more <activity> child elements intended to be executed		

2.6 Runtime Environment

To execute CSCL scripts written in ELFML a script engine is required. As it can be seen on Figure 2-3, the runtime environment is composed of an interpreter and a middleware, which are installed and executed in both teacher's and students' devices. The script engine interpreter is composed by an XML parser and a plug-in manager. The XML parser reads the ELFML scripts, and validates their syntax according to the XML schema that defines the ELFML language. On the other hand, the plug-in manager loads and registers plug-ins included by the scripts, allowing the interpreter to instantiate the classes provided by them.

It is possible to develop third-party plug-ins for use with ELFML scripts through APIs provided by the middleware, namely, the Control and Environment APIs. These APIs define standard interfaces and interaction protocols that are implemented by the plug-ins. The Control API contains the necessary abstractions for developing Actions and Operators, while the Environment API provides the necessary base classes and interfaces for implementing Tools, UI Controls and Artifacts (e.g. a custom document format).

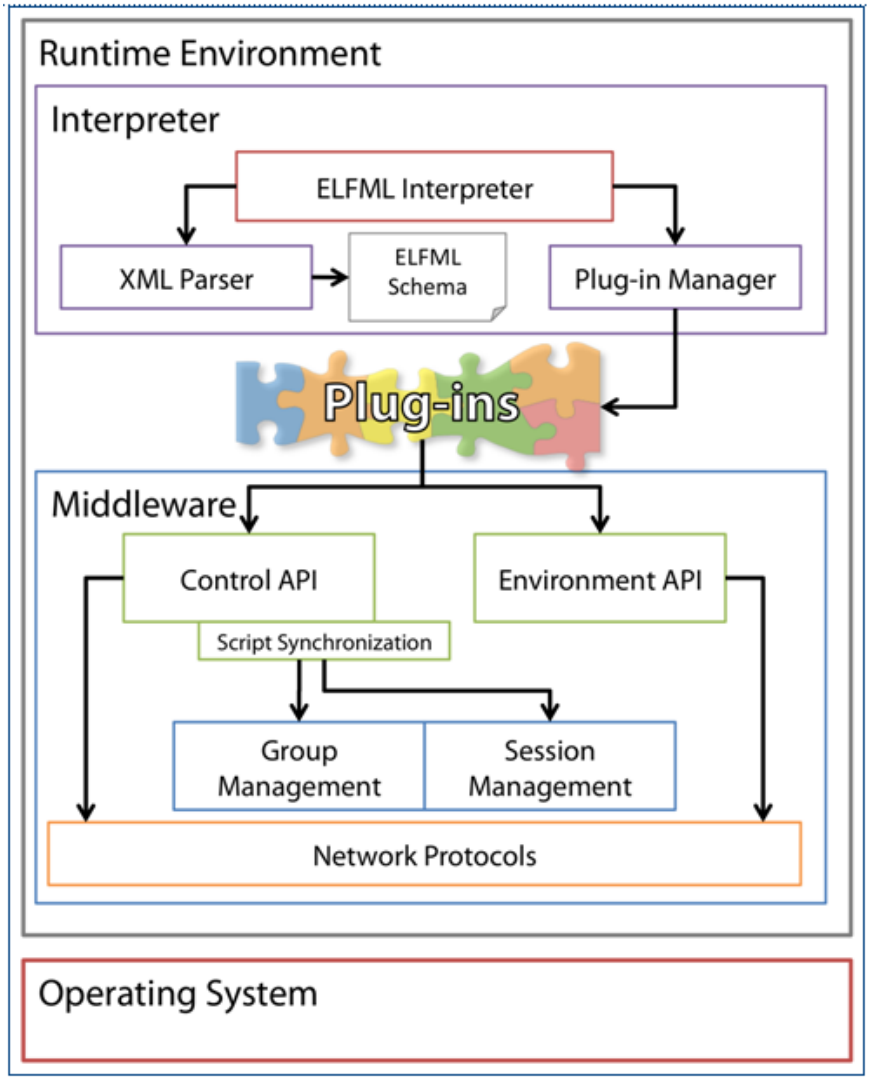
The middleware mediates between the operating system and the interpreter, playing three major roles at runtime:

Coordination and synchronization of students' script execution: The middleware provides a protocol for synchronizing and coordinating execution of scripts on the student devices, ensuring that the script flow is maintained consistently in the participating devices at all times.

Session and group management: Different classrooms can share the same IP network and operate in isolation of each other through establishing different sessions. The middleware allows the teacher establishing the sessions and students joining them. During a session, one or more scripts can be run sequentially, and student groups can be flexibly created and modified during execution of each script.

Inter-device communications: Text message, file and binary data transfers over TCP or UDP multicast protocols. The middleware can reestablish TCP connections automatically whenever they fail.

Figure 2-3 – Runtime Environment Architecture

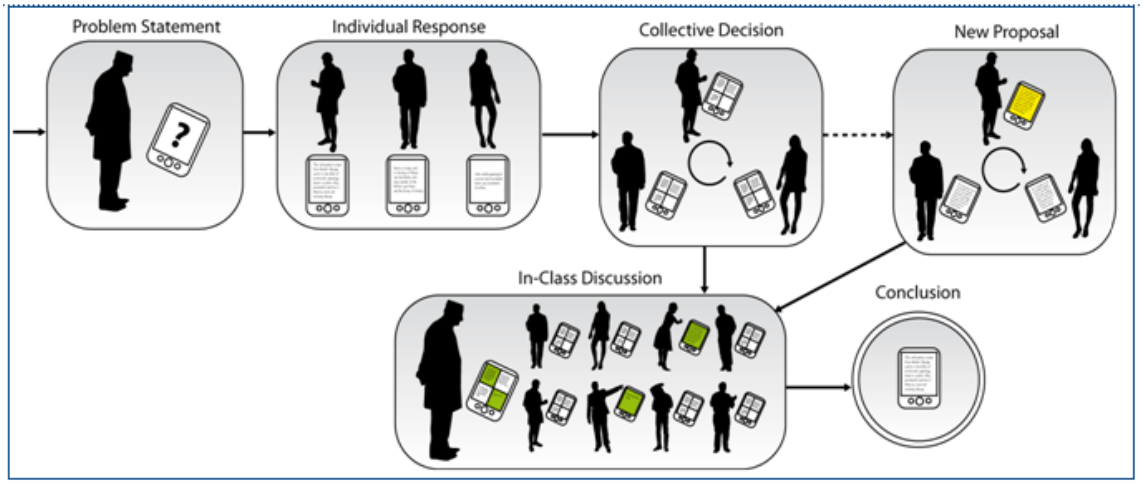


2.7 A Case Study: CollPad

Using our technology we developed an MCSCL application based on the CollPad script (Nussbaum et al., 2008a). The development process lasted for six weeks and was carried out by a team composed by a product manager, an education researcher, an instructional designer and a software developer. In the following sections we describe the CollPad script, the process by which it was implemented and the results we obtained by using the presented technology.

2.7.1 Description

Figure 2-4 - CollPad Script



The CollPad script is a problem resolution process in which students must solve a problem stated by the teacher, through three successive phases involving individual work, group discussion and classroom discussion, respectively. The teacher and the students participate in the script with wirelessly interconnected PDAs equipped with a touch screen (e.g. Pocket PC). Figure 2-4 shows the script flow of CollPad. When the script starts, each student sees the group he/she has been assigned to on his/her device's screen. Once all students have met in their respective groups, the teacher sends them a problem (Problem Statement phase). In the Individual Response phase, each student handwrites an answer to the teacher's problem in the device's touch screen, without interacting with others. Once all members of a group submit their individual answers, the group jumps to the Collective Decision phase.

In the Collective Decision phase, students compare and discuss their individual answers, coming to the decision whether they submit to the teacher one of the responses they wrote already, or propose a new one. If all students agree to choose the same answer, the answer is automatically submitted to the teacher, and they wait for the In-Class Discussion phase. Otherwise, the system prompts them to come to an agreement. If the students agree to write a new answer, the group jumps to the New Proposal phase. In this phase, the script randomly assigns the scribe role to one of the students and the reviewer role to his/her companions. When the scribe writes a new response, the reviewers must agree on accepting or rejecting it. If the reviewers accept the new response, it is then submitted to the teacher and the group waits for the In-Class Discussion phase. Otherwise, if all the reviewers disagree with the new answer, the system randomly picks a new scribe and the process is repeated.

To begin the In-Class Discussion, the teacher selects a set of answers to his/her question submitted by the groups in the Collective Decision or New Proposal phases of the script, to be matter of the discussion. The teacher may call for a vote on the answers he/she selected, and pick random students (defenders) from each group, who must defend the answer submitted by their respective groups verbally. In this way, all the different answers involved in the discussion are analyzed and debated by the students with the teacher's mediation, until the whole classroom agrees on a final response to the original problem (Conclusion).

2.7.2 Implementation

Relying on the description of the CollPad script, the development team jointly elaborated a prototype of the software, which consisted of a complete user interface mockup. By means of the prototype, the developer determined the environment elements of the script (e.g. tools, UI controls, artifacts for visually representing questions and answers) and inferred the pertaining activities, rules and roles, groups and artifacts required for creating a complete flow diagram of the script. Relying on the flow diagram and the environment elements defined, the software developer elaborated a list containing all the classes necessary for instancing the meta-model of Figure 2-2.

For implementing CollPad, it was necessary to write ELFML code to represent the entire script, develop all the action classes, tools (e.g. handwriting-based virtual notepad), artifacts (e.g. handwriting document) and some of the user interface classes (e.g. a viewer for displaying multiple handwriting documents in thumbnail size). The development of these classes was necessary due to the inexistence of previous implementations that could be reused.

The player we used to run CollPad is based on the Edunova Network middleware described by Echeverria et al. (2006), which is entirely written in Microsoft Visual C++ 2005, and has been specially designed for implementing MCSCL applications in the Microsoft Windows Mobile platform. We decided to rely on the Edunova Network middleware because of its maturity and trustworthiness, with more than four years of development and deployment in schools (Galloway, 2007; SRI, 2008). The middleware provided our player with robust communications, coordination and synchronization services over WiFi (802.11b) networks in ad-hoc and infrastructure modes.

On Table 2-2 we show a breakdown of the efforts required for implementing CollPad. The effort was divided in writing the ELFML script description and in developing the meta-model classes required in C++. The leftmost column shows the items that needed to be implemented, and the following columns show the resulting amount of lines of code in the software and the person-hours that were dedicated to coding, testing, debugging and optimizing the application. The total effort for implementing CollPad ascended to 274 person-hours, that is, about six person-weeks, which is a marginal effort in comparison to the near 3000 hours (18 person-months) that took to implement the player, including the Edunova Network middleware. The effort could have been lessened about a 30% were it had been possible to reuse Artifact, Tool and Action classes from other scripts.

Table 2-2 - CollPad Implementation Effort Breakdown.

Item	Qty.	Lines of Code (LOC) Count	Coding (Person-Hours)	Laboratory Testing (Person-Hours)	Debugging and Code Refactoring (Person-Hours)
ELFML Script	1	1600 (XML)	20	4	8
Action Classes	28	2840 (C++)	64	20	20
User Interface Classes	3	1970 (C++)	60	12	18
Tool Classes	1	1020 (C++)	20	8	4
Artifact Classes	2	520 (C++)	12	2	2
Total	35	1600 (XML) and 6350 (C++)	176	46	52
	Total (Person-Hours)	274			

Comparatively, developing Actions required less effort than developing UI Controls and Tools. This was because the latter needed to be developed from scratch using the graphics programming libraries from the Microsoft MFC application framework, which resulted particularly error-prone and complex for the developer, mainly due to C++ memory issues (Heine and Lam, 2003), such as memory leaks, dereferencing invalid pointers, etc. On the other hand, the development of Actions was based on the Control API of the middleware, which easily allowed taking full advantage of the middleware features. Compared to the number of user interface classes, the amount of actions is greater because these implement a substantial portion of the application logic in a highly modularized way.

The specification of the script in ELFML was a transversal activity through the development process and was done incrementally, as the necessary script plug-ins and classes became available. For writing ELFML code, the developer used a text-based XML editor that provided syntax validation through the ELFML XML schema.

2.7.3 Obtained Results

The meta-model established by the framework and its instantiation through the scripting language facilitated composing activities, environments, the script flow, as well as assembling these components altogether. The custom meta-classes created for CollPad were able of taking full advantage of the functionality and services provided by the framework's bottom layers, such as network protocols and script synchronization mechanisms. As a result, the instantiation of the meta-model saved time and effort in the development process, because it allowed the developer focusing on the aspects strictly concerned with the script. Consequently, it was not necessary to define an entirely new architecture for the application nor it was required to address lower level concerns, such as network programming.

We tested the CollPad script implemented with our framework in three experiments; the first one was conducted in three schools located in Wolverhampton, U.K, involving five 6th grade teachers for a month (Galloway, 2007), the second one was conducted in Chile with three math high school teachers in two schools, also for a month (Nussbaum et al., 2008a), and the last experiment was held in the Knowledge Management and Human-Computer Interface courses lectured at the Computer Science Department of the School of Engineering at Pontificia Universidad Catolica de Chile, during the fall and spring semesters of 2007, respectively (Nussbaum et al., 2008b). All the experiments used Windows Mobile-based Pocket PCs; Fujitsu-Siemens Pocket Loox 720 and N650 in the U.K, and HP iPaq rx1950 and Dell Axim X50 devices in Chile. The networks were based on WiFi (802.11b) in infrastructure mode.

In the British schools, 30 to 35 students participated in each CollPad run, while in Chile, class groups enrolled 30 to 40 students. In the courses held at Pontificia Universidad Catolica de Chile, CollPad was executed a total of 21 times, one time per lesson, with 20 to 25 students participating in each script run. From the performance standpoint, the script engine based on the Edunova Network middleware (Echeverria et al., 2006) yielded very satisfactory results regarding the script loading time, responsiveness of the user interface, and network reliability and scalability with a large number of devices. The software was non-disruptive, easy to setup and use in the classroom, and therefore well adopted by teachers and students alike.

2.8 Discussion

As model-driven engineering advocacy claims, domain-specific languages and tools can narrow down the conceptual gap between the problem and the implementation domains of discourse, and therefore, reduce the complexity of developing software (France and Rumpe, 2007). We believe this view certainly applies to the development of CSCL software, given the strong necessity of multidisciplinary collaboration that exists in designing and implementing this kind of applications (Hernandez-Leo et al., 2006). In the domain of CSCL as well as in MCSCL, the problem-implementation gap described by France and Rumpe (2007) arises when developers implement software solutions to problems using abstractions that are at a lower level than those used by non-technical practitioners (e.g. education researchers, instructional designers, etc.) to express the problem, leading to accidental complexities that make software development difficult and costly.

Having considered a theoretical formalization of CSCL scripts in the specification of our scripting language, we can narrow the problem-implementation gap, because both the meta-model and the scripting language empower developers with abstractions and discourse situated at the problem-level. Nonetheless, at this early stage of development, our technology is only appropriate for technical experts as it lacks proper tools that could enable its use by non-technical practitioners, who could largely increase its applicability and adoption. Even so, the framework we have developed reaches beyond solutions that focus on overcoming the technical challenges found in mobile networks for collaborative activities (Echeverria et al.,

2006), because it leverages a domain-specific meta-model to facilitate complying with the requirements of MCSCL applications.

During recent years, Web-standards based technologies, such as the Rich Internet Applications (Blom et al., 2008) and process execution languages like BPEL (Ouyang et al., 2005), have emerged to facilitate developing platform-independent collaborative applications. However, these applications are heavily dependent on distributed components and locked into client-server architectures, with intensive use of communication networks, as the business logic and back-end services are typically implemented in remote servers. While these technologies may be suitable for distributed and/or asynchronous CSCL applications, they are difficult to adopt in synchronous, "Web-unplugged" and server-less collaboration environments like our technology targets (Echeverria et al., 2006).

For the future development of our script engines we consider that the best alternative is sticking to fat-client technologies, such as Java ME or Microsoft .NET Compact Framework (Blom et al., 2008). These are based on virtual machines that solve software portability problems, and enhance the productivity of software development with features such as garbage collection and state of the art development tools (code editors, debuggers, etc.). Although virtual machines can have a diminishing effect on performance (Echeverria et al., 2006), we believe that the increasing capacities of mobile devices will be able to compensate this drawback in the future.

2.9 Conclusions and Future Work

CSCL scripts permit modeling and describing the forms in which technology supports collaborative learning scenarios, upon the basis of meta-models and high-abstraction languages specific to the problem domain (France and Rumpe, 2007). This characteristic allows developing software in less time and at a lower cost. Nevertheless, it requires undertaking the initial investment of developing script engines able to interpret the script instructions subject to the limitations and technical restrictions introduced by mobile devices. In this article, we presented a framework based on CSCL scripts that facilitates the development of applications for face-to-face collaboration environments, supported by wirelessly networked handheld devices, such as PDAs and smartphones.

In a face-to-face collaboration environment, mobile devices are beneficial for simulating social interactions in order to elicit fruitful collaboration and consequently, produce better learning outcomes (Cortez et al., 2005; Liu et al., 2007; Zurita et al., 2003). However, taking advantage of the mobility and low invasiveness qualities of mobile devices brings difficulties associated with the strong limitations and technical restrictions of the devices. In spite of the difficulties, our findings show that wirelessly networked mobile devices are capable of operationalizing CSCL scripts in an autonomous way, without need of full-fledged servers or sophisticated network infrastructure. CSCL scripts are therefore a promising approach for facilitating the development of experiments and applications in the field of MCSCL. Although developing a script engine for MCSCL environments

is costly, developing interpreted scripts is indeed more economical and flexible than developing an ad-hoc application for each pedagogical design.

By adopting the "Applications = Components + Scripts" guiding principle (Nierstrasz and Achermann, 2000), our framework promotes developing flexible software that can be tailored and customized to suit new learning scenarios. Nevertheless, this does not assure that script components of a particular script can be easily reused in other scripts, or that script components can be modified to comply with new requirements. We believe that a substantial step towards facilitating software reusability will be accomplished if we focus on supporting generalisability (Dillenbourg, 2002). We could work towards this in the future by providing plug-in libraries and script templates to support recurrent patterns found in collaborative learning scenarios, in the form of collaborative learning flow patterns (CLFPs) (Bote-Lorenzo et al., 2004; Hernandez-leo et al., 2006), or script schemata as defined by Dillenbourg & Jermann (2006).

In order to facilitate the adoption of our technology, we will focus in the near future on developing visual authoring tools for designing CSCL scripts, with the capability of automatically generating ELFML code, and in developing a platform-independent script engine that enables creating easily portable applications.

Bibliography

Adams, D. & Hamm, M. (1996). *Cooperative learning: critical thinking and collaboration across the curriculum.* Springfield, IL: Thomas Publisher Published by Charles C Thomas Pub Ltd.

Aïmeur, E., Frasson, C., Lalonde, M. The Role of Conflicts in the Learning Process. *SIGCUE OUTLOOK 12*, 27 (2), 2001.

Aronson, E., & Patnoe, S. (1997). *The jigsaw classroom: building cooperation in the classroom.* (2 ed.). United States: Addison-Wesley Educational Publishers Inc.

Baines E., Blatchford P. & Kutnick, P. (2003) Changes in grouping practices over primary and secondary school. *International Journal of Educational Research* 39, 9–34.

Belotti, V. & Bly, S. (1996). Walking away from the desktop computer: distributed collaboration and mobility in a product design team. *Proceedings of the 1996 ACM conference on Computer Supported Cooperative Work*, USA, 96, 209–218.

Blom, S., Book, M., Gruhn, V., Hrushchak, R., Kohler, A. (2008). Write Once, Run Anywhere A Survey of Mobile Runtime Environments. *In: Grid and Pervasive*

Computing Workshops, 2008. GPC Workshops '08. The 3rd International Conference, 25-28 May 2008, pp. 132 – 137, Kunming, China.

Bote-Lorenzo, M. L., Hernández-Leo, D., Dimitriadis, Y. A., Asensio-Pérez, J. I., Gómez-Sánchez, E., Vega-Gorgojo, G., and Vaquero-González, L. M. (2004). Towards Reusability and Tailorability in Collaborative Learning Systems using IMS-LD and Grid Services. *Advanced Technology for Learning*, 1(3), 129 – 138.

Caeiro. M., Anido, L., and Llamas, M. (2003). A Critical Analysis of IMS Learning Design. In B. Wasson, S. Ludvigsen & U. Hoppe (Eds), *Proceedings of the International Conference on Computer Support for Collaborative Learning*, Dordrecht: Kluwer Academic Publishers, 2003, 363 – 367.

Caeiro-Rodriguez, M., Llamas-Nistal, M., and Anido-Rifon, L. (2007). A Separation of Concerns Proposal for the Modeling of Collaborative Learning. *International Conference on Convergence Information Technology*, 21-23 Nov. 2007, pp. 368 – 375.

Cortez, C., Nussbaum, M., López, X., Rodríguez, P., Santelices, R., Rosas, R., and Marianov, V. (2005). Teachers' support with ad-hoc collaborative networks. *Journal of Computer Assisted Learning*, 21(3), 171 – 180.

Cortez, C., Nussbaum, M., Santelices, R., Rodríguez, P., Zurita, G., Correa, M., and Cautivo, R. (2004). Teaching Science with Mobile Computer Supported Collaborative Learning (MCSCL). *2nd IEEE International Workshop on Wireless and Mobile Technologies in Education (WMTE'04)*, pp. 67 – 74.

Crook C. (1994) Computers and the Collaborative Experience of Learning. Routledge, London, UK.

Dillenbourg, P. (1999). What do you mean by collaborative learning?. In P. Dillenbourg (Ed) *Collaborative-learning: Cognitive and Computational Approaches*, Oxford: Elsevier, 1 – 19.

Dillenbourg P. (2002). Over-scripting CSCL: The risks of blending collaborative learning with instructional design. *In P. A. Kirschner (Ed.), Three worlds of CSCL Can we support CSCL*, pp. 61–91, Nederland: Heerlen, Open Universiteit.

Dillenbourg, P., & Jermann, P. (2006). Designing integrative scripts. In: F. Fischer, I. Kollar, H. Mandl, & J. Haake (Eds.), *Scripting computer-supported collaborative learning: Cognitive, computational and educational perspectives*. New York: Springer.

Dillenbourg, P., Hong, F. (2008). The mechanics of CSCL macro scripts. International *Journal of Computer-Supported Collaborative Learning*, 3(1), 5 – 23. Springer New York.

Dillenbourg, P., and Tchounikine, P. (2007). Flexibility in macro-scripts for computer-supported collaborative learning. *Journal of Computer Assisted Learning*, 23(1), 1 – 13.

Dan, Y., and XinMeng, C. (2007). Creating Computer Supported Collaborative Learning Activities with IMS LD. *Human-Computer Interaction. HCI Applications and Services*, pp. 391 – 400. Springer Berlin / Heidelberg.

Dan, Y., Zhang, W., Chen, X. (2006). New Generation of E-Learning Technologies. First International Multi-Symposiums on Computer and Computational Sciences - Volume 2 (IMSCCS'06), pp. 455 – 459.

Echeverria, S., Santelices, R., and Nussbaum, M. (2006). Comparative Analysis of Ad-Hoc Networks Oriented to Collaborative Activities. *In Architecture of Computing Systems - ARCS 2006: 19th International Conference, Frankfurt/Main, Germany*, March 13-16, 2006. Proceedings, volume 3894 of Lecture Notes in Computer Science, 2006, pp. 465 – 479, Springer.

Ellis, C. A., Gibbs, S. & Rein, G. L. (1991). Groupware: some issues and experiences. *Communications of the ACM*, 34(1), 38–58.

Engestrom, Y. (1987). Learning by expanding: an activity-theoretical approach to developmental research. Orienta-Kosultit, Helsinki, Finland.

Fischer, F., Kollar, I., Haake, J.M., Mandl, H. (2007). Introduction. In: F. Fischer, I. Kollar, H. Mandl, & J. Haake (Eds.), *Scripting computer-supported collaborative learning: Cognitive, computational and educational perspectives*. New York: Springer.

France, R., and Rumpe, B. (2007). Model-driven Development of Complex Software: A Research Roadmap. In Future of SE Track of ICSE'07, 2007, pp. 37 – 55.

Galton M., Hargreaves L., Comber C., Wall D. & Pell T. (1999) Changes in Patterns of Teacher Interaction in Primary Classrooms: 1976-96. *British Educational Research Journal*, 25(1), 23–37.

Galloway, J. (2007). When three is not a crowd The Guardian of London (UK), Education supplement. June 19. Available at: http://education.guardian.co.uk/elearning/story/0,,2105774,00.html [accessed October 3, 2008].

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Griffiths, D., Blat, J., García, R., Vogten, H., and Kwong, K.L. (2005). Learning Design tools. In Koper, R. & Tattersall, C. (Eds.), *Learning Design, a Handbook on Modelling and Delivering Networked Education and Training*. Springer Heidelberg, pp. 109 – 135.

Haake, J. and Pfister, H. (2007). Flexible Scripting in Net-Based Learning Groups. *Scripting Computer-Supported Collaborative Learning. Cognitive, Computational and Educational Perspectives*. Fischer, Kollar, Mandl, Haake (Ed.), pp 155 – 175. Springer.

Heine, D. L., and Lam, M. S. (2003) A practical flow-sensitive and context-sensitive C and C++ memory leak detector. *In PLDI*, pp. 168–181.

Hernández-Leo, D., Burgos, D., Tattersall, C., and Koper, R. (2007). Representing CSCL macro-scripts using IMS LD: Lessons learned. Available at: http://dspace.ou.nl/bitstream/1820/784/7/hernandezleo_etal_creation-scripts.pdf [accessed October 3, 2008].

Hernández-Leo, D., Villasclaras-Fernández, E. D., Asensio-Pérez, J. I., Dimitriadis, Y., Jorrín-Abellán, I. M., Ruiz-Requies, I., and Rubia-Avi, B. (2006). COLLAGE: A collaborative Learning Design editor based on patterns. *Educational Technology & Society*, 9(1), 58 – 71.

IMS Learning Design Specification v1.0 (2003). Available at http://www.imsglobal.org/learningdesign/index.html [accessed October 3, 2008]

Johnson, D. & Johnson, R. (1999). Learning Together and Alone. Cooperative, Competitive, and Individualistic Learning. Publiser Allyn.

Jurado, F., Redondo, M. A., and Ortega, M. (2006). Specifying Collaborative Tasks of a CSCL Environment with IMS-LD. *Cooperative Design, Visualization, and Engineering*, 311 – 317. Springer Berlin.

Kobbe, L., Weinberger, A., Dillenbourg, P., Harrer, A., Hämäläinen, R., Häkkinen, P., and Fischer, F. (2007). Specifying computer-supported collaboration scripts. *International Journal of Computer-Supported Collaborative Learning*, 2(2-3), 211 – 224. Springer New York.

Kollar, I., Fischer, F., and Slotta, J.D. (2005). Internal and external collaboration scripts in web-based science learning at schools. In T. Koschmann, D. Suthers & T. W. Chan (Eds.), *Computer-Supported Collaborative Learning 2005: The next 10 years!*, pp. 331 – 340. Mahwah, NJ: Lawrence Erlbaum.

Lave J. and Wenger E. (1991). Situated Learning. Cambridge, England: Cambridge University Press.

Liu, C., Tao, S., Ho, K., Liu, B., and Hsu, C. (2007). Constructing an MCSCL Groupware to Improve the Problem-solving Experience of Mathematics for Hearing-impaired Students. *Seventh IEEE International Conference on Advanced Learning Technologies (ICALT 2007)*, pp. 345 – 347.

Martens, H., and Vogten, H. (2005). A Reference Implementation of a Learning Design Engine. *Learning Design* (pp. 91 – 108). Springer Berlin / Heidelberg.

Miao, Y., Harrer, A., Hoeksema, K., and Hoppe, H. (2007). Modeling CSCL Scripts - A Reflection on Learning Design Approaches. In: F. Fischer, I. Kollar, H. Mandl, & J. Haake (Eds.), *Scripting computer-supported collaborative learning: Cognitive, computational and educational perspectives*. New York: Springer.

Miller, S. M. (2002). Vygotsky and education: the sociocultural genesis of dialogic thinking in classroom contexts for open-forum literature discussions. Retrieved September 29, 2008, from Hanover College, Psychology Department Website: http://psych.hanover.edu/vygotsky/miller.html.

Nierstrasz, O., and Achermann, F. (2000). Supporting Compositional Styles for Software Evolution. *In Proceedings International Symposium on Principles of Software Evolution (ISPSE 2000)*, Nov 1-2, 2000, Kanazawa, Japan, pp. 11 – 19.

Nussbaum, M., Alvarez, C., McFarlane, A., Gomez, F., Claro, S., and Radovic, D. (2008a) .Technology as small group face-to-face Collaborative Scaffolding. *Computers & Education*, doi:10.1016/j.compedu.2008.07.005.

O'Donnell, A. M., & Dansereau, D. F. (1992). Scripted cooperation in student dyads: A method for analyzing and enhancing academic learning and performance. *In R. Hertz-Lazarowitz and N. Miller (Eds.), Interaction in cooperative groups: The theoretical anatomy of group learning* (pp. 120- 141). London: Cambridge University Press.

Ouyang, C., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M., and Verbeek, H.M.W. (2005). Formal Semantics and Analysis of Control Flow in WS-BPEL. Technical Report BPM-05-15. BPM Center.

Rogoff B. & Lave J. (1984) Everyday cognition: Its development in social context. Cambridge, MA: Harvard University Press.

Shachar H. & Sharan S. (1994) Talking, relating, and achieving: Effects of cooperative learning and whole- class instruction. *Cognition and Instruction* 12(4), 313–353.

Shen, C., Lesh, N., Moghaddam, B., Beardsley, P. & Bardsley, R. S. (2001). Personal digital historian: user interface design. *Design expo: CHI '01 extended abstracts on human factors in computer systems* (pp. 378–385). Seattle, WA.

Silverman, B.: Computer Supported Collaborative Learning (CSCL). *Computers & Education*, Vol. 25 No. 3 (1995) 81-91

SRI (2008). SRI International's Evaluation of TechPALS Software Shows Promising Results in Teaching Fractions. Available at: http://biz.yahoo.com/iw/080611/0405564.html [accessed October 3, 2008].

Suthers, D. (2007). Roles of Computational Scripts. In: F. Fischer, I. Kollar, H. Mandl, & J. Haake (Eds.), *Scripting computer-supported collaborative learning: Cognitive, computational and educational perspectives*. New York: Springer.

Szyperski, C. (2003). Component technology – what, where, and how?. *Proc. of the 25 th Intl. Conf. on Software Engineering (ICSE)* (pp. 684–693). Portland, OR, USA.

Tremainel, M., Sarcevic, A., Wu1, D., Velez, M., & Bogdan (2005) Size Does Matter in Computer Collaboration: Heterogeneous Platform Effects on Human-Human Interaction, *Proceedings of the 38th Hawai International Conference on System Sciences* - 2005

Uden, L. (2007). Activity theory for designing mobile learning. *Int. J. Mobile Learning and organization*, 1(1), 81 – 102.

van Engelen, R. (2004). Code generation techniques for developing light-weight xml web services for embedded devices. *In SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pp. 854 – 861, ACM Press.

Vavoula, G., and Karagiannidis, C. (2005). Designing Mobile Learning Experiences. *In P. Bozanis and E.N. Houstis (Eds.): PCI 2005, LNCS 3746, 2005*, pp.534- 544, Springer-Verlag, Berlin Heildelberg.

WFMC (2008) Workflow Management Coalition. Available at http://wfmc.org [accessed October 3, 2008].

Yin, C., Ogata, H., and Yano, Y. (2007). Participatory simulation framework to support learning computer science. *International Journal of Mobile Learning and Organi*zation, 1(3), 288 – 304.

Zurita, G., and Nussbaum, M. (2004a). Computer supported collaborative learning using wirelessly interconnected handheld computers. *Computers & Education*, 42(3), 289 – 314.

Zurita, G., and Nussbaum, M. (2004b). A constructivist mobile learning environment supported by a wireless handheld network. *Journal of Computer Assisted Learning*, 20(4), 235 – 243.

Zurita, G., and Nussbaum, M. (2007). A conceptual framework based on Activity Theory for mobile CSCL. *British Journal of Educational Technology*, 38(2), 211 – 235.

Zurita, G., Nussbaum, M., and Salinas, R. (2005). Dynamic Grouping in Collaborative Learning Supported by Wireless Handhelds. *Educational Technology & Society*, 8(3), 149–161.

Zurita, G., Nussbaum, M., and Sharples, M. (2003). Encouraging face-to-face collaborative learning through the use of handheld computers in the classroom. Human Computer Interaction with Mobile Devices and Services. *Springer Verlag Lecture Notes in Computer Science*, vol. 2795, 2003, pp. 193–208. Springer Berlin.

Appendices

Appendix A: Reception and confirmation email

From: J. Michael Spector
Subject: **RE: Article submitted to ETRD, confirmation request.**
Date: 15 de octubre de 2008 19:00:17 GMT-03:00
To: Claudio Álvarez <cjalvare@ing.puc.cl>
Cc: JungMi Lee <jl06c@fsu.edu>

Received ...

Dear Dr. Spector and JungMi:

The following manuscript has been received by Educational Technology Research & Development:

Manuscript Number: ETRD-283

Title: A framework to facilitate the development of face-to-face CSCL applications supported by wirelessly networked mobile devices

Author(s): Mr. Claudio Javier Alvarez, Rosa A Alarcon, Ph.D.; Miguel Nussbaum, Ph.D.

This manuscript has been submitted to the Development Section.

J. Michael Spector, Ph.D., Prof. & Assoc. Director, Learning Systems Institute
Florida State University, C 4622 University Center
Tallahassee, FL 32306-2540 USA
TEL +1 850 645 1777 / FAX +1 850 644 4952

-----Original Message-----
From: Claudio Álvarez [mailto:cjalvare@ing.puc.cl]
Sent: Tuesday, October 14, 2008 11:45 PM
To: mspector@lsi.fsu.edu
Subject: Article submitted to ETRD, confirmation request.

Dear Professor Spector,

I'm Claudio Alvarez, a M.Sc. student from the School of Engineering at PUC Chile, and I've just submitted an article to the ETRD journal (development area) through the online process. The article is titled "A framework to facilitate the development of face-to-face CSCL applications supported by wirelessly networked mobile devices". Could you please send me an e-mail message confirming that the article was properly submitted? I need your confirmation to comply with administrative requirements from the Graduate Studies department at my school. Thank you very much in advance for your comprehension.

Best regards,

Claudio Alvarez

Appendix B: ELFML Language Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema targetNamespace="http://xml.elf.org/schema/ELFML"
```

```
    xmlns:elf="http://xml.elf.org/schema/ELFML"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="script" minOccurs="1" maxOccurs="1">
 <xs:complexType>
  <xs:choice>
    <xs:element ref="elf:activity" minOccurs="0" maxOccurs="unbounded" />
    <xs:element ref="elf:environment" minOccurs="0" maxOccurs="unbounded" />
    <xs:element ref="elf:include" minOccurs="0" maxOccurs="unbounded" />
    <xs:element ref="elf:meta" minOccurs="0" maxOccurs="unbounded" />
    <xs:element ref="elf:student" minOccurs="0" maxOccurs="unbounded" />
    <xs:element ref="elf:teacher" minOccurs="0" maxOccurs="unbounded" />
  </xs:choice>
   <xs:attribute name="id" type="xs:ID" use="required" />
   <xs:attribute name="strictELF" type="xs:boolean" default="true" />
 </xs:complexType>
</xs:element>

<xs:element name="meta">
 <xs:complexType>
   <xs:attribute name="key" type="xs:string" use="required" />
   <xs:attribute name="value" type="xs:string" use="required" />
 </xs:complexType>
</xs:element>

<xs:element name="include">
```

```
 <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="elf:plugin" />
      <xs:element ref="elf:file" />
   </xs:choice>
 </xs:complexType>
</xs:element>

<xs:element name="plugin">
 <xs:complexType>
    <xs:attribute name="source" type="xs:string" use="required" />
    <xs:attribute name="name" type="xs:string" use="required" />
 </xs:complexType>
</xs:element>

<xs:element name="file">
 <xs:complexType>
    <xs:attribute name="source" type="xs:string" />
 </xs:complexType>
</xs:element>

<xs:element name="activity">
 <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="elf:action" />
```

```
        <xs:element ref="elf:actionseq" />
        <xs:element ref="elf:operator" />
        <xs:element ref="elf:variable" />
        <xs:element ref="elf:groupConfiguration" />
        <xs:element ref="elf:eventListener" />
    </xs:choice>
    <xs:attribute name="id" type="xs:ID" use="required" />
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="environment" type="xs:string" default="none" />
    <xs:attribute name="entrypoint" type="xs:boolean" default="false" />
  </xs:complexType>
</xs:element>

<xs:element name="environment">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="elf:uiControl" />
      <xs:element ref="elf:tool" />
      <xs:element ref="elf:contentUnit" />
    </xs:choice>
    <xs:attribute name="id" type="xs:ID" use="required" />
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="base" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
```

```
<xs:element name="actionseq">
 <xs:complexType>
  <xs:sequence>
     <xs:element ref="elf:action" minOccurs="1" maxOccurs="unbounded" />
  </xs:sequence>
    <xs:attribute name="id" type="xs:ID" />
    <xs:attribute name="name" type="xs:string" />
    <xs:attribute name="activationMode" type="xs:string" default="manual">
   <xs:simpleType>
       <xs:restriction base="xs:string">
         <xs:enumeration value="automatic" />
         <xs:enumeration value="manual" />
     </xs:restriction>
   </xs:simpleType>
    </xs:attribute>
 </xs:complexType>
</xs:element>


<xs:element name="action">
 <xs:complexType>
  <xs:sequence>
     <xs:element ref="elf:param" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
    <xs:attribute name="id" type="xs:ID" />
```

```
    <xs:attribute name="name" type="xs:string" />
    <xs:attribute name="pluginId" type="xs:string" default="engine" />
    <xs:attribute name="class" type="xs:string" use="required" />
    <xs:attribute name="activationMode" type="xs:string" default="manual">
   <xs:simpleType>
       <xs:restriction base="xs:string">
         <xs:enumeration value="automatic" />
         <xs:enumeration value="manual" />
     </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
</xs:element>

<xs:element name="teacher">
 <xs:complexType>
  <xs:sequence>
     <xs:element ref="elf:activity" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="student">
 <xs:complexType>
  <xs:sequence>
```

```
      <xs:element ref="elf:activity" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="uiControl">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="elf:param" minOccurs="0" maxOccurs="unbounded" />
      <xs:element ref="elf:rect" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required" />
    <xs:attribute name="pluginId" type="xs:string" default="engine" />
    <xs:attribute name="class" type="xs:string" use="required" />
    <xs:attribute name="layer" type="xs:integer" default="1" />
  </xs:complexType>
</xs:element>

<xs:element name="tool">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="elf:param" minOccurs="0" maxOccurs="unbounded" />
      <xs:element ref="elf:rect" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required" />
```

```
    <xs:attribute name="pluginId" type="xs:string" default="engine" />
    <xs:attribute name="class" type="xs:string" use="required" />
    <xs:attribute name="layer" type="xs:integer" default="1" />
 </xs:complexType>
</xs:element>

<xs:element name="contentUnit">
 <xs:complexType>
  <xs:sequence>
     <xs:element ref="elf:param" minOccurs="0" maxOccurs="unbounded" />
     <xs:element ref="elf:rect" minOccurs="1" maxOccurs="1" />
  </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required" />
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="source" type="xs:string" />
    <xs:attribute name="layer" type="xs:integer" default="1" />
    <xs:attribute name="renderer" type="xs:string" use="required">
   <xs:simpleType>
       <xs:restriction base="xs:string">
         <xs:enumeration value="html" />
         <xs:enumeration value="custom" />
    </xs:restriction>
   </xs:simpleType>
          </xs:attribute>
 </xs:complexType>
```

```
</xs:element>

<xs:element name="eventListener">
 <xs:complexType>
  <xs:sequence>
     <xs:element ref="elf:param" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required" />
    <xs:attribute name="eventName" type="xs:string" use="required" />
    <xs:attribute name="activationMode" type="xs:string" default="activitystart">
   <xs:simpleType>
       <xs:restriction base="xs:string">
         <xs:enumeration value="activitystart" />
         <xs:enumeration value="always" />
     </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
</xs:element>

<xs:element name="groupConfiguration">
 <xs:complexType>
  <xs:sequence>
     <xs:element ref="elf:param" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
```

```
    <xs:attribute name="id" type="xs:ID" use="required" />
    <xs:attribute name="name" type="xs:string" />
    <xs:attribute name="pluginId" type="xs:string" default="engine" />
    <xs:attribute name="class" type="xs:string" use="required" />
    <xs:attribute name="policy" type="xs:string" default="default" />
  </xs:complexType>
</xs:element>

<xs:element name="param">
  <xs:complexType>
    <xs:attribute name="key" type="xs:string" use="required" />
    <xs:attribute name="value" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>

<xs:element name="variable">
  <xs:complexType>
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="value" type="xs:string" use="required" />
    <xs:attribute name="type" type="xs:string" use="required">
     <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="string" />
          <xs:enumeration value="boolean" />
          <xs:enumeration value="integer" />
```

```
            <xs:enumeration value="float" />
            <xs:enumeration value="expression" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

 <xs:element name="operator">
  <xs:complexType>
     <xs:attribute name="id" type="xs:ID" use="required" />
     <xs:attribute name="name" type="xs:string" use="required" />
     <xs:attribute name="class" type="xs:string" use="required" />
     <xs:attribute name="pluginId" type="xs:string" default="engine" />
  </xs:complexType>
 </xs:element>
</xs:schema
```