Improving Understanding of Distributed Systems Through Shared Work Among Teacher and Students

Justo N. HIDALGO
Department of Computer Science and Engineering, University Antonio de Nebrija, c/Pirineos, 55, MADRID, jhidalgo@nebrija.es

KEYWORDS*: distributed systems, case-based lecture, teacher participation, industrial-based project, mentoring*

ABSTRACT: *Teaching distributed systems can not only based upon theoretical facts. The use of design and development projects which make use of different methodologies and technologies, such as UML, RPC, CORBA or RMI, are invaluable tools for the student to understand what a distributed application is. However, these are usually short-period projects, in which no more than three or four students are involved. This paper presents a experience of how a deeper involvement of the teacher when designing and developing a case-based project can dramatically improve students' capabilities of understanding the theoretical concepts underneath a distributed system, and, consequently, the quality of the systems created. Students have worked, along with the teacher, in a Technology Watch project which allowed us to implement different architecture: Peer-to-peer, MOM and Client/Server. The teacher has participated both as a coach and as just another designer and developer, so that (1) students always have a structured way to teamwork, but, (2) they are forced to actually solve each of the challenges that an industrial-based project imposes. Mentoring has as main advantages the control of the project throughout the semester, and monitoring the effort of each of the individuals. Students are forced to come up with the answers to each of the problems the group can face, as the teacher will not usually want to produce the correct one. This leads to a decision-making process, always desired in last-year courses. The paper is structured as follows: firstly, it will show how classic methodologies do not fit in this kind of subjects. An explanation of the subject methodology comes later, along with a detailed definition of the project developed during this academic year. Results and consequences are shown, which prove how students come up with much better results, and with a much higher degree of satisfaction.*

1 INTRODUCTION

Teaching Distributed Systems in a Software Engineering title must take into account both the theoretical side -showing the students how a Network Operating System is composed, RPC semantics, coordination strategies, and so on- and a practical one in which the student can get more involved, applying what they are learning in class to an implemented exercise. In the last years, due to an improvement on machine facilities and labs, teachers in most universities are creating a set of labs (mainly two or three) which have to be solved by the student either by themselves or working in teams. Having the students work on design and programming labs has been a major improvement on this subject, and helps them to be more prepared.

Nevertheless, we believe this is not enough for a student to be ready for industry. A student must achieve a degree of excellence far above what we are used to now. Building distributed systems has become one of the most demanding activities for Software Engineers, due to the unstoppable advance of internet-based applications. The basis of J2EE (Java 2 Standard Edition) [1], Microsoft .NET, and DCOM technologies [2], or LAMP (Linux-Apache-MySQL-PHP) programming [3] lies on the accurate knowledge of how these systems are built, that is, exactly what a Distributed Systems subject should teach -such as, for example, Sockets, RPC, RMI, CORBA ORBs, and such-.

Besides, this subject is usually found in one of the last two years of engineering, which means that the student is about to come out to the "real world" -or even is already there, if they have had some

internship-. Building a software system has much to do with designing and programming, but it has also to do with working in teams, understanding what the client wants from the beginning, applying a good methodology, controlling pressure and stress, ... Even though this is not a Software Engineering subject, students must apply multidisciplinary skills once they get this far in their courses [4]. In the following sections these factors are explained in terms of this particular subject.

1.1. Teamwork

Short-period projects or labs are usually built either individually or in small teams -2, 3 students at most-. Even though many current real-world projects are developed in small teams: (1) student team members are usually friends, used to working together, which does not usually happens in industry, and (2) labs are usually described by the teacher in a completely non-ambiguous way, and with a purely academic goal. This is necessary, but not sufficient for a complete understanding of the context. Teams just have to "get the idea" of what the teacher wrote in the assignment, build it, try it, and deliver it.

1.2. Decision-making process

As said above, the lab is a concrete, unambiguous task which, most of the times, does not need any further explanation, more than maybe thirty minutes of class time. The design of the solution is implicitly shown, and even an API (application programmer's interface) is defined so that the teacher can apply a set of automatic tests for every single delivery. This type of assignments helps the students to formalize their work, to understand the concept of interface and systems integration, and also helps the teacher to check their work in an easier way. Nevertheless, the students have no chance to make any true decision more than those related to algorithm-programming. The path is already defined by the instructor.

1.3. Pressure

Every assignment in an engineering title must be planned so that the students have enough time to handle it in a correct way. The teacher must take into account how long their course lasts to create different assignments, which can be mandatory or optional, and that this chore will be done in a particular context, that is, that students have other courses with more labs and activities.

The students know that if the teacher tells them that they have three weeks to deliver a particular assignment, the assignment is actually thought to be perfectly done in that period of time. Is this typical in real-world projects?

1.4. Mentoring

Students in modern universities have access to a wide range of resources, from which they can access to useful information for their courses and individual quest for knowledge. Their primary resource must always be the teacher, which helps them through the different options for each problem. Sadly, most of the times the students see their teachers as masters of a particular area, and try to follow their advices and apply them accurately to their assignments. There is a big difference between guidance and mentoring. Last two years of engineering should focus on teaching the students that for each problem there usually are many different solutions, where the best ones of them depend on the context.[5]

As it has been shown in this introduction, modern approaches to teaching distributed systems lack real-world challenges. Creating different small-size projects as assignments helps the students to understand most of the theory, but is not enough. To overcome this problem, this paper proposes the use of a modified version of the "case method", typically used in MBAs and Business Schools, where not only a particular case –or example- is shown to the students, but also, where the whole class must work together, and where the teacher acts as both a mentor and a worker, avoiding to be a "problem-solver" [6].

The following chapter will detail how this methodology is defined, showing how the Distributed Systems subject in University Antonio de Nebrija comprises all of the challenges shown above.

2 TOPICS OF THE SUBJECT

"AT4118:Distributed Systems" is an optional subject for second-cycle students –4th and 5th year- of Computer Science and Engineering of the University Antonio de Nebrija, Madrid, in Spain. It is divided in twenty-six 1h20min-sessions. The main topics of the subjects are the following:

1. Introduction to Distributed Systems

What is a distributed system? What is the difference between a distributed system and a parallel system? And among all questions, what are the characteristics that a distributed system should possess? A through process explaining the basic concepts behind concurrency, fault tolerance, load distribution, transparency, and so on, is executed.

2. Distributed Systems Architecture

Because this subject is mainly focused on the design of distributed applications, it is very important for the students to know about the most relevant architectural patterns, such as Layers or Broker, and the most important classes of systems that can be built: Client/Server, Peer-to-peer, and MOM (asynchronous messaging) structures.

3. Network Operating System concepts

The Layers pattern can then be broadened by studying the Microkernel architecture, and understanding how advanced services can be created and dynamically deployed over basic functionality. Even though Microkernel implementations have not had enough interest for industry to be successfully produced, their ideas have been very useful for distributed architectures' creation. Later on, the concept of middleware is explained.

4. Communications Service

Of course, the basic concept of a distributed system is "communication", so it has to be studied in detail. Starting with a summary of IPCs and shared-memory concepts from Operating Systems, and Sockets from Network Communications, this topic focuses on RPC (Remote Procedure Calls) and MOM (Message-Oriented Middleware). RPC is studied thoroughly, from its interface processing through IDLs to its different semantics. CORBA ORB, Java RMI and .NET basics are explained too.

5. Other services

The most important services are also explained, such as Naming and Directory services (with examples of DNS, X.500 and LDAP), Distributed File systems (NFS) or Security service (p.e. Kerberos).

The topics are typical in a subject of this kind in any computer-related Engineering [7] [8]. To improve understanding of these areas, students must work in small teams to design and implement three different, but related, distributed systems:

1. The first assignment is the implementation, using Java Sockets, of a mediator-based peer-to-peer application –such as Napster-. The development with sockets allows students to see how difficult can be for them to code every single characteristic of a distributed system without any help, and how much time professional developers can lose when they can not focus on the business logic.
2. The second assignment introduces the use of Java RMI to the students, asking them to implement a Client/Client-Server/Server application to study characteristics such as fault tolerance, load balancing, and allow the students to investigate advanced uses of Java remote interfaces.
3. The third and last assignment also uses RMI to implement a MOM system, using a Publish-Subscribe architectural pattern. It also serves as an introduction to serialization –the messages sent to the MOM server must be stored- and activation framework –the MOM server and publishers are waken up whenever a new subscription message arrives-. It also introduces RMI/IIOP as a different CORBA-based implementation.

3 CHALLENGES

The subject, with the characteristics shown in the previous chapter, has been working for the last four years, with minor adjustments. Even though it has always been of great interest to the students, and many students have obtained their first job mainly because of the knowledge and practice obtained in this subject, and closer and self-critic study of the methodology brings out some challenges that are not adequately solved.

First of all, each assignment has a predefined architecture, which the students must implement. Even though the teacher does not impose it –the team could come up with a way of using another one-, it is very difficult for that to happen, for the system to design is very small.The size of the assignment also imposes that only a single architecture is implemented, so students can not integrate different subsystems.

In third place, the students must not even think about the feasibility of the project: is it possible to implement it? Is it possible to finish it before the deadline? These are very significant aspects of every project in industry that are not taken into account in this subject.

Another important challenge is the choice of which characteristics should a particular project have. Fault tolerance, load balancing, or different types of transparency imply much work and many difficulties that we must study before starting its implementation. Assignments explicitly explain the students what to do, and what not to do. This is necessary in order to avoid ambiguities.

As it has been said above, the use of a single architecture prevents the students from designing adequate and well-defined APIs so that other applications can plug into it, and communicate, thus allowing for openness.

Other minor challenges not covered in this subject are the following:

- The programming language is imposed. No previous study of what language is better suited for a particular poblem is accomplished.
- The goal is to get the lab running. Average-to-low level students just care about providing an implementation which “just works”.

All of these challenges can be solved with the methodology shown above, but it does not come out naturally – it would mean forcing the students to think of a mini-project as a real-world application-.

4 PROPOSED METHODOLOGY

The methodology proposed by this paper is strongly influenced by the case method, but with some important differences, mainly because of the diferent targets of its use: prepared for undergraduate students of a technical course with little or no professional experience. The case method, originally developed by Harvard University, is currently used by most Business Schools; the main characteristic is the use of companies’ cases –absolutely real or adapted-, told by the instructor and followed by a discussion among them and students, about the circumstances around it. This method provides the class a methodically-led discourse that, when adequately implemented, allows the students to make the best decisions and improve their know-how.

As it has been said above, this subject could not take exactly this method, for the topic and the target were completely different from what Harvard had in mind when they created this idea. In the following paragraphs the concepts behind this version are explained.

The basic idea is to force the students to work on the theoretical concepts from the beginning of the academic year. To achieve this goal, a class project was introduced right at the beginning of the second chapter – so they have already learnt about Introduction to Distributed Systems and the basics of Architecture of Distributed Systems-, in which a case is described, asking them to provide a software product at the end of the semester, which accomplishes the whole set of requirements of the original document. Instead of creating teams which had to implement the same lab, the class is divided in teams, each of which will have to take care of one subsystem of the whole application. The decomposition of the project in subsystems is also realized by the whole class, with a little help of the teacher, reasoning from

the characteristics of the distributed systems previously explained, and by the basic ideas about architecture –Layers, N-tier architecture, and so on-.

From that moment, each session is planned as follows: the course's web page has a document for each topic which tells everything that the student needs for each session –actually, it is planned that the set of documents becomes a book once it gets more mature-; students must have read the document's related paragraph for that particular session. The first twenty minutes of class are used to answer any question the students may have, and to focus on any aspect that the teacher can consider, either because of the students' questions, or because of the document's lack of clarity. The following hour is used to work on the case method-based project –which will be explained in the following section-.

The project is led by a waterfall life cycle –see figure 1-, with the following phases:

1. Requirements: the first two sessions must be used by the students to ask any question refering what the product has to do. They will have an proposal document which will explain them the initial idea, but the teacher, acting as a client, must answer all of the questions that might help to reduce any incongruity. It is important for the teacher to differentiate between incongruities and ambiguities. This last concept is at the very bottom of the idea of case method, and should not be removed: students must cope with them, and make their own decisions to cancel them.
2. Design: right after the first questions have been answered by the "customer", students must start to make decisions about what subsystems will the software development have. Once a consensus has been reached, teams are created, each of which will be responsible for one subsystem, and its relationship with the rest. Depending on the group, an "interface leader" can be chosen, who will be the one who will talk to the other "interfaces", and will decide how each subsystem will communicate with the others. Meanwhile, the rest of the team will start to work on its internals, and decide what type/s of architecture is/are the most convenient. Design is loosely based on UML, but heavily relies on object-oriented techniques, such as interface-and-delegation-based design and patterns.
3. Implementation: after the design decisions have been made, and the interfaces are clear, the class must decide what programming language to use. Of course, undergraduate students tend to choose the language they know better, so the teacher must help them recognize the advantages and flaws of using each possible one. After the decision is taken, each team starts to code its part.
4. Test: testing is divided in three parts, being them unit testing, subsystem testing and system integration testing. This last one is realized by the interface leaders of each team, which, at the design phase should have created a set of test cases for each different relationship between two subsystems.
5. Deployment: non-functional requirements must be taken into account to prepare the installation process.

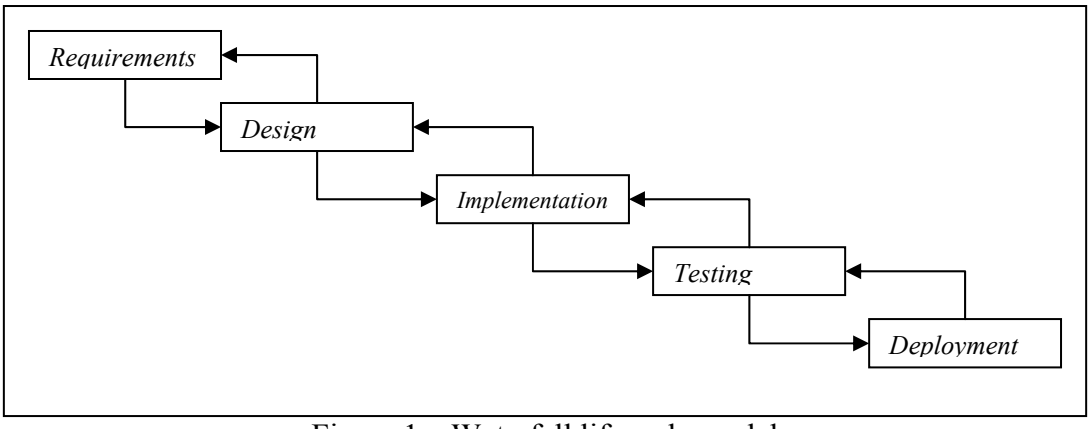

Figura 1 – Waterfall lifecycle model

The whole work is made in the classroom during session time. No homework is mandatory, mainly because students will also have to code three assignments, such as the ones explained in section 2. But every single artifact they produce –documentation, notes, code, test cases, ...- is stored in a collaborative environment, administered by the teacher. This way, every component of the project has access to every single piece of information that may require during classtime or afterwards.

5 CASE DESCRIPTION

This section will summarize the main characteristics of the project. As the project is based on Competitive Intelligence, a small introduction to that concept will be given first, followed by the description of the tool to be built by the students.

5.1 Competitive Intelligence Basics

Organizations must always show its capacity for solving problems and challenges in the most professional way. To achieve these goals, they must obtain all kinds of information, produced both internally and by external entities, such as competitors, public entities – bibliography, research articles, web links, and so on-. In some cases, the difficulty resides on finding enough information; but, nowadays, the great challenge is different: there is too much data on several resources such as libraries, information-related companies and, above all, the internet –through its services such as world-wide-web, news servers, or FTP-. The task of choosing the apropriate information from the correct data sources is called Technology Watch. When this process also involves analysis of data, and a set of proposed actions to execute, the concept of Competitive Intelligence arises. [9] y [10] are good initial sources for technology watch and competitive intelligence definitions and concepts.

5.2 Description of the Tool

The introduction to the project is the following: “You are part of a development team of a small company based in Competitive Intelligence. Your short-term goal is just to find a niche market, so you don’t need big technologic solutions, because the reports to be delivered to your clients will be mainly written by infonomists, experts in that particular sector.

Nevertheless, the time spent by infonomists is currently too much, for they have to search for information in a manual way. The main source of information is the web, an non-structured and difficult-to-search environment. The main tool used by infonomists is Google, a search robot, but that implies a few problems:

1. Each infonomist must be very careful when choosing keywords for a particular query; a bad decision in this moment and bad results will be returned by the search engine.
2. Searches must be repeated continuously, because Google updates its registries and rankings at a very fast pace.
3. Repeated searches return many replicated results. When reports are built by a distributed team of infonomists, the task can be very difficult to manage.
4. If things go well in the company, infonomists could work in different sectors, so a single search could apply to several of them.

The project aims for the construction of a peer-to-peer competitive intelligence tool which can help overcome these difficulties.”

Summarizing, the basic requirements of the tool are:

- Each infonomist will have access to one or more sectors.
- Each infonomist will be able to define the keywords from which they wish to find some information. These keyword will have to be thoroughly studied before using them.
- The infonomist will then configure the search frequency, that is, the rate in which their tool will search the web using the Google API.

- The results are stored either locally or in a server –not specified by the requirements, so the students can discuss about it: this discussion can lead to a client/server architecture, a P2P, or a hybrid one-.
- The system must check which results are duplicated from previous search from the same or other infonomist –how to implement this functionality will depend on what type of architecture has been decided; this will influence APIs between subsystems-.
- The infonomist will select a web source to study –a non-automatic task-: each source will have a particular state. If a source is "Clear", it has never been checked by any infonomist, so it will be chosen by the worker, and marked as "In Study". If a source is "In Study", some other infonomist is in charge. If the state is "Valid", some infonomist has checked the source and decided that is valid for that particular sector –or subsector-; current infonomist can now decide to check some other source, or "recheck" this one, changing its state to "In Study". Finally, if the source's state is "Rejected", the infonomist can decide to recheck it too.
- If a source is in a "Valid" state, it will also have a subjective classification between 1 and 10, meaning the interest this source has in relation with the sector it is included.
- The system must work even though some infonomists have not got their applications started.
- It is supposed that many infonomists work at home.

As it can be seen, the project is based on a real world situation, with real restrictions and many ambiguities that students must overcome. There is some detail in a few parts of the description –such as the state transitions-, and little or none in some others –p.e. will the infonomist access the system from a web-based interface, or with a front-end?-. As we will see in the following section, allowing the students to decide what is best for each challenge must be carefully managed by the teacher in one of its roles in this subject: mentoring.

6 MENTORING AND DEVELOPING

The teacher has different roles in this subject. First of all, their classic role, explaining the theory, is canceled, obliging the students to study by their own, and asking every question they had in the following session. This way of acting is typical in Ph.D. programs, but second-cycle students should have no problem with it. The teacher then becomes both mentor and part of the project team.

6.1 Mentor

At the beginning of the project, the teacher must be very careful of how the project is growing up. The role of the mentor is divided in the following tasks:

Project presentation: presenting the project is one of the most important parts of the subject, and many precautions must be taken in order to avoid giving too much information to the students. The teacher must act as a client who is actually expecting the teams to provide the solutions, making their own decisions, as long as that means something good for the future tool. The other risk the teacher must avoid is the opposite: the fear of too much information provokes a lack of vital data which lets the students with a sense of ambiguity that is too strong, and the antipattern "Analysis paralysis" appears.

The mentor must also be the creator of the first version of the project. Even though it is not mandatory, the fact that the mentor shows that the tool can be done, that it is not an impossible task, can help the most conservative students to realize of their own potential. In this course's case, a small RMI-coded Client-Server application which was able to use the Google API to return a query result was enough to get the students going –actually, this first version was implemented and tested in less than three hours-.

Choosing the subsystems for the project must be led by the mentor, even though only two goals must be in their mind: each subsystem should have aproximately the same degree of complexity, and each

subsystem should have at least some sense, and be useful for the whole system. The rest should be up to the students. The teacher must also remember that the goal of this project is not actually how well coded it will finally be, because that characteristic will be checked and corrected in the small labs and assignments.

The mentor must also monitor the improvements of each subsystem, and force the students to apply the information and knowledge from each chapter as they go through them.

6.2 Developer

The previous section shows that, basically, the teacher is acting as a project manager. But this paper also proposes that students get more involved in the project if they actually see their teacher cooperating with them at the same level, that is: designing and developing with them, in the classroom. The teacher must be a part of different subsystems, so that:

- the whole class can learn from the teacher's experience.
- the whole class will lose the fear of committing mistakes, because, probably, the teacher will have their own too.
- the teacher can help the less able students, by pair-programming with them.
- at the opposite side, the most gifted students can be challenged by the teacher to create more complex structures or more advanced functionality, but also learning to encompass it with the need of respecting the interfaces among subsystems.

How the teacher works with every team will depend on how many people there are. This year, as there were not many teams, the teacher was able to work with all of them in the same session.

7 ROLE OF THE STUDENTS

The goal of a second-cycle student can not be just to pass the exam and obtain a grade. Students must behave in a proactive way, always looking for further information. Their role in this particular subject is divided in different tasks: as it has been explained above, each student must read each section before its related session so that any question can be asked. The whole class, acting together as a whole project team, will be responsible of choosing the set of subsystems in which the application will be divided, as well as the relationship among them. Finally, each subsystem will be designed and coded by a group of students, as detailed before. Table 1 shows the different activities a subsystem team must do.

Table 1. Subsystem team's list of activities

Activity	Description
External interface definition	Definition of the interfaces –APIs- that will be used by other subsystems to communicate with it.
Detailed requirements eliciting	Meeting with the client and studying the initial documentation to clearly define what the subsystem must accomplish.
Definition of boundary limits	Explicit definition of what the subsystem must and must not do, and to what level.
Election of types or architectures to use	Decision of which basic architectures –C/S, P2P, MOM- or hybrid ones form the baseline of the subsystem.
Design of the subsystem	The object-oriented design of the subsystem must be achieved by the team.
Description of test cases for system integration testing.	Definition of every test case necessary to verify and validate the integration among their subsystem and the rest of them.
Implementation of the subsystem	Coding in the selected programming language of the previous design. Unit testing is included.
Testing	Realization of system and integration testing.

8 CONCLUSIONS AND FUTURE WORK

This methodology has been used in the Distributed Systems subject in course 2003/04. After finishing the semester, the following conclusions have been obtained:

1. The student gets more involved in the subject, applying the theoretical concepts to a real-world-based project, and from the beginning.
2. The teacher helps the students to avoid a lack of confidence because of initial fear due to the size of the project. An initial version of the tool provided by the mentor makes them believe it is possible.
3. The academic success seems to be also very interesting. 75% of the students passed the exam, a much better result than in the previous five years. Also, in a subjective analysis, students finished the subject with a much higher level of understanding, and better design skills.
4. Students were able to choose among different types of architecture for a particular problem, and to apply them properly using common industry standards, such as the Layers or Broker pattern. Some of the results produced were of industry-level quality.
5. Just as a side effect, because of working in an industry-based project, the students' resume is broadly benefited by this subject.

Some drawbacks and improvements define the future work to be done:

- Use of iterative and incremental software development process. The waterfall model does not adequately mimics current software development standards such as the Unified Process or eXtreme Programming. This first year the waterfall model was used because it was not found how to apply iterative models in such a short period of time. Further study must be used to find a solution.
- Improvement of the project plan. Even though the requirement elicitation and design processes were properly produced during the semester, not much code was obtained because of lack of time. The teacher tended to spend too much time on these two first phases, just as it usually happens in industry when a new methodology is introduced. Better use of mentoring must be achieved so that the whole process is adequately deployed.
- Also, the project plan must be improved to map more effectively with the theoretical sessions. Table 2 details what it was expected at the beginning of the course, and the percentage of completion.
- Creation of a case pool: even though the Competitive Intelligence case will be used in following years, it could be interesting to develop a set of different cases to use alternatively, just as Business Schools do.
- Finally, this methodology will be used in some other subjects, such as Software Engineering, Internet Systems and Software Quality.

The use of this methodology means much more work to the teacher, as it has to produce all of the information in a text-based format, create a very complete project description, and apply all of the mentoring in an adequate way. But, as it has been discussed, the improvement over a more classical master class is radical, and we plan to keep on using it in the future.

Table 2. Initial project planning expectations, and percentage of completion

Topic	Project planning	% of completion
Distributed Systems Architecture	Initial analysis of the application. Division of application in subsystems. Study of the most valid architectures for each module.	100%
Network Operating Systems Concepts	Use and need of Microkernel pattern. Definition of API for each subsystem.	100%
Communications Service	Detailed definition of architecture for each subsystem. Serialization and activation framework. Implementation.	75% (just basic implementation)
Other services: naming service	Design and implementation Utilization of an LDAP service.	75% (design and basic implementation)
Other services: distributed file system	Concept of single-system image. Use of a DFS.	0%
Other services: event service	Implementation of MOM server.	25% (basic design)

ACKNOWLEDGEMENTS

The author would like to thank the students Angel Luengo, Juan Sobrino and Diego Pardilla for their help and effort during this experience. He would also like to thank Jose Luis Delgado, director of the subject's knowledge area, for believing in this project.

REFERENCES

[1] Sun Java 2 Enterprise Edition. Available from web: <URL: java.sun.com/j2ee>

[2] Microsoft .NET Distributed Technology. Available from web: <URL: www.microsoft.com/net>

[3] ONLAMP: O'Reilly resources about LAMP technologies. Available from web: <URL: www.onlamp.com>

[4] ELLIS, H. J. C., MEAD, N. R., MORENO, A. M., SEIDMAN, S. B., 2003. *Industry/University Software Engineering Collaborations for the Successful Reeducation of Non-Software Professionals.* Proceedings of the 16th Conference on Software Engineering Education and Training (CSEET'03), pp. 44-51. 1093-0175/03

[5] GNATZ, M., LEONID, K., PRILMEIER, F., SEIFERT, T. 2003. *A Practical Approach of Teaching Software Engineering.* Proceedings of the 16th Conference on Software Engineering Education and Training (CSEET'03), pp. 120-128. 1093-0175/03

[6] RAJU, P.K., SANKAR, C. S. 1997. *Teaching Real-World Issues in Engineering Classrooms Through Case Studies*, Thomas C. Evans Instructional Unit Award Lecture, ASEE Southeastern Conference, Atlanta, GA., April 1997

[7] COULOURIS, G., DOLLIMORE, J., KINDBERG, T. 2000. Distributed Systems: Concepts and Design. Addison-Wesley Pub, 3rd. Edition. 0201619180.

[8] ALLAMARAJU, S. et al. 2001. Professional Java Server Programming J2EE, 1.3 Edition. Wrox Press Inc., 1st. Edition. 1861005377.

[9] VEDDER, G., VANECEK, M. T., GUYNES, C. S., CAPPEL, J. J. 1999. *CEO and CIO Perspectived on Competitive Intelligence*, Communications of the ACM, 1999

[10]MALHOTRA, Y. 1996. *Competitive Intelligence Programs: An Overview*. @BRINT Research Institute. Available from web: <URL: www.brint.com/papers/ciover.htm>