From Programming to Modeling: Our Experience with a Distributed Software Engineering Course

Jordi Cabot
Universitat Oberta de Catalunya
Barcelona, Spain
jcabot@uoc.edu

Francisco Durán, Nathalie Moreno, Antonio Vallecillo
Universidad de Málaga
Málaga, Spain
{duran,vergara,av}@lcc.uma.es

José Raúl Romero
Universidad de Córdoba
Córdoba, Spain
jrromero@uco.es

ABSTRACT
Distributed Software Engineering (DSE) concepts in Computer Science (or Engineering) Degrees are commonly introduced using a hands-on approach mainly consisting of teaching a particular distributed and component-based technology platform (such as Java Enterprise Edition or Microsoft .NET) and proposing the students to develop a small distributed software application with it. Though this approach provides the students with some relevant practical knowledge, we believe that it is not the most appropriate way of teaching all the concepts and particularities of DSE. Thus, in this paper we report on our experience of redesigning an initial DSE course following a model-based approach. By raising the level of abstraction we gained modularity, separation of concerns and technology independence, while making the course evolve according to the latest trends in software development methods.

Categories and Subject Descriptors
D.2.0 [**Software Engineering**]: General – *standards.*
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *object-oriented design methods.*
D.2.11 [**Software Engineering**]: Software architectures – *domain-specific architectures, data abstraction, patterns.*
D.2.12 [**Software Engineering**]: Interoperability – *distributed objects, interface definition languages.*

General Terms
Design, Experimentation, Standardization, Languages.

Keywords: Education; Distributed Software Engineering; Virtual University; Component-based Software Development; UML; ODP

1. INTRODUCTION
Software Engineering (SE) is primarily an engineering discipline, and therefore SE Education should focus more on the principles, basic concepts and high-level models of the systems than on the particularities and technical issues of programming languages and technology platforms used to implement them. Although this is progressively being achieved by the majority of most SE curricula courses, some subjects are still too focused on the (low-level) programming details and the complexity of the existing implementation platforms. One representative example can be found in Distributed Software Engineering (DSE), understood as the engineering of distributed software [1] and not as the process of distributed development of software. Probably because of the inherent complexity of most distributed object and component platforms, and the large number of aspects that need to be considered in a distributed application, most of the DSE courses focus on the description of the concepts and mechanisms of one particular platform. Therefore, the students are provided with a whole set of complex and low-level concepts, most of them specific to the particular platform being taught, without receiving a global view of the software architecture of the system and even without learning the means to represent the different high-level models of the system that capture the separate aspects involved in its development, distribution and deployment. In other words, in most DSE courses the trees do not allow the students to see the forest.

At the same time, the growing adoption of Model-Driven Development (MDD) and Model-Driven Architecture (MDA) approaches in today's software development projects is also shifting the focus of existing SE methods from code to (higher-level) models, which have now become the primary artifacts of the software development process.

Taking these two ideas into account, it looks like there is a clear need to make our traditional DSE courses evolve in order to raise their level of abstraction, but without losing their practical nature. This trade-off between teaching high-level DSE concepts and mechanisms in a platform-independent manner, while at the same time providing a hands-on approach to developing distributed applications on at least one platform is one of the current challenges of teaching DSE.

In this sense, the main goal of this paper is to present our experience with the re-design of a DSE course taught at the Open University of Catalonia (UOC) [2], a fully virtual university founded in 1993 and with more than 5.000 computer science students. In the UOC, the DSE course, which is taught during the first course at the Master's degree, is the most advanced of all SE related subjects. In the previous courses, students learn the phases of the software development life-cycle and how to specify and design (using UML) a given software system. Fundamental concepts of distributed systems (such as asynchronous communications or distributed clocks) are also taught in previous courses. Then, the DSE course addresses the specificities of

ICSE'08, May 10–18, 2008, Leipzig, Germany.

developing distributed software systems with emphasis on the use of architectural and software components during the specification and implementation of such distributed systems. The DSE course has an estimated workload of 6 ECTS credits [3] (around 180 work hours in total).

This paper discusses the educational context in the UOC (Section 2), the previous DSE course design (Section 3), the reasons that motivated us to change the course (Section 4), our new targets and objectives (Section 5) and the design of the new course (Section 6). Finally, we present some preliminary conclusions after the completion of two terms with the new design (Section 7). In summary, we show how it is possible to reconcile both a global view of a distributed system and a high-level definition of the DSE concepts and mechanisms, with a hands-on and practical approach to DSE. However, our experience has not been free from challenges and difficulties, which need to be shared so that the rest of the SE educational community can benefit from our experience and from the lessons that we have learnt.

2. (E-)LEARNING DSE IN THE UOC

The UOC is a fully virtual university founded in 1993. Currently, it has more than 35.000 students, 5.000 of them enrolled in the Computer Science degree. Virtual universities have appeared as an alternative (or a complement) to conventional (also called traditional or face-to-face) ones, especially for those students unable to physically attend face-to-face classes.

The main difference with respect to conventional universities lies in the learning process, and consequently, on how students acquire the knowledge and skills required in the course. It shifts from having the teacher as its main source to the combination of materials, consultants, other students and the virtual environment. This change in the learning process poses additional challenges during the preparation of a DSE course for the UOC and, in general, for any kind of virtual institution. In what follows, some of theses issues are discussed.

The first thing to consider is the characteristics of the virtual environment. In a virtual university, the student becomes part of a virtual campus that provides a learning environment as well as other kinds of services. For each course, the student finds a virtual class (see Figure.1) led by a consultant teacher. Like conventional classrooms, a virtual class in the UOC includes a communication space, with a message board where the consultant posts important information to the whole class and a forum where students can exchange ideas with both the consultant and the other students. Nevertheless, virtual classes go even further. They also include a planning space, a resources space and an assessment space. The planning space includes information about the course (objectives, description, assessment method, etc.) along with instructions to follow it properly (scheduling when to study each course unit, deadlines for assessment activities, etc.). The resources space groups all the course contents, including the teaching materials (online version), additional materials (such as complementary exercises or past exams), links to software resources (as CASE tools), case studies and a recommended bibliography available by means of the virtual library. The assessment space records the students' assignments and their marks.

To succeed in this virtual environment, students must be able to develop abilities such as autonomy, discipline and self-organization (there are no schedules; each student must manage his/her own learning process individually). Proactive attitude and the capability to acquire and share information with other students are also required in order to carry out collaborative work (especially relevant in all SE courses to overcome semantic relativism issues and to obtain a unified solution). All these abilities require student maturity, and therefore virtual courses are usually discouraged for young students (as an example, 89% of the UOC students are between 25 and 55 years old and are employed). To help students organize their learning, for each course a detailed schedule must be provided. The schedule of our DSE course contains not only details about the exercises and tests to be done during the term, but also (weekly) recommendations about how to distribute the course contents throughout the term.

It is also very important to provide the students with adequate tutoring support during their studies In the virtual model, the face-to-face teacher is replaced by a team of teachers, each one with a well-defined role: the consultant (in charge of the student learning a particular subject), the tutor (permanent mentoring support during all of the degree course) and the UOC staff teacher (coordination of the other teacher roles and in charge of the course contents and organization). See [4] for further information on these roles.

The main task of the consultant teacher is to facilitate the student learning process by clarifying concepts, answering their questions and motivating them in the context of a specific course. The consultant is also in charge of the student assessment. A consultant of a DSE course should not be in charge of more than 50 students. This has proven to be the upper limit for a quality learning experience. Moreover, we should benefit from the fact that a virtual model is an asynchronous time and space model (no fixed schedule is required) where the consultants hired can come both from the professional and academic world. We have observed that having professional software developers as teachers of the DSE course is a key issue in motivating the students, since they see more easily the practical application of all contents.

To facilitate the student/teacher interaction, a UOC policy declares that all questions must be answered in a 24-hour timeframe. Moreover, we encourage the use of the forum as the best communication channel, instead of private email exchange between tutors and students. In this way, all students can benefit from (and participate in) the discussions between the consultant and the students. This is especially helpful when trying to reach a consensus between the students and the consultant during the more creative tasks of the software development process.

Another important factor in e-learning courses is the quality of the teaching material (which must partially replace conventional courses lecturers). Being the primary reference for students, the quality of materials directly affects the quality of the course. Materials should be specifically designed to facilitate the virtual learning of the course content. Although this is a lengthy process (the development of the materials for this DSE course took us one year), student learning is easier than when using existing materials or books as a primary reference. Additional materials (such as case studies or frequently asked questions) can be released in the virtual campus anytime.

Finally, due to the number and complexity of the tools used in the DSE course, we found it necessary to provide additional support in

the form of virtual labs. A virtual lab is a special kind of virtual class with a specific consultant and materials where the student can solve questions about the technical problems derived from the use of tools (installation, configuration, usage, etc.). Ideally, the consultant of a virtual lab should be a (recent) former student since they are the most likely to know the common problems students will face during the course. Materials for the virtual lab (installation manuals, FAQs, etc.) must be kept up to date with the latest tool releases. We have realized that even small changes between the manual and the tool setup greatly confuse the students and generate many questions in the virtual class forum.

Figure 1. Partial view of a virtual class for the DSE course, showing the planning space (*Planificación* in Spanish), with the course information (*Plan Docente*) and the scheduling (*Calendario*).

3. FORMER DSE COURSE DESIGN

The previous DSE course was clearly oriented to learning the Java Enterprise Edition (JEE) platform. The main goal was for students to be able to program a small distributed software application using JEE technologies (mainly Java Server Pages and Entity Beans), to understand the role of the different parts of the three-layer application and how those parts communicate with each other.

In this sense, the contents of the DSE course consisted of three different modules. The first one provided a brief introduction to distributed systems and a description of the RMI, CORBA and DCOM technologies. The second module was devoted to a study of the component technology, including some basic definitions and their representation in UML. Finally, the third module (and by far the largest one) contained a thorough description of the different technologies forming the JEE platform.

The course was organized so that students completed the first two modules during the first five weeks of the course (1/3 of the term). For each module a small assessment activity was proposed mainly consisting of five short questions on basic concepts of distributed systems (module 1) and a couple of specifications of UML component diagrams (module 2).

The rest of the term (10 weeks aprox.) was devoted to studying the JEE platform and to the development of the small application commented above. Students unable to provide a fully functional and error-free application at the end of the term automatically fail the course (regardless of the marks obtained in the first two activities).

4. REASONS FOR A CHANGE

Clearly, considering the contents described in the previous section, that course was more of a programming course than a software engineering course. The course was more focused on describing a particular technology than on explaining concepts and techniques common to all kinds of distributed systems. After a few terms, we realized that this focus had a number of drawbacks that hindered the correct learning of the DSE concepts. Examples of such drawbacks include:

- Students were unable to develop complex systems. This requires raising the level of abstraction far beyond their programming-oriented view.
- No methodological aspects about the specification/design of distributed software systems were taught, and the few ones that did appear in the course were mostly hidden by the technical details of the JEE platform, considered to be more relevant by the students.
- There was an important lack of (software) architectural concepts. As a result, we found that students always proposed the same solution (the classical 3-tier architecture) to all kinds of distributed systems presented during the course.
- The focus on specific technologies caused that many of the contents to become quickly outdated due to the fast pace of technology evolution. This implied that teaching materials needed to be continuously updated and, more importantly, that part of the time invested by the students in the course became obsolete shortly afterwards. Instead, a more abstract view of the topic and concepts is more stable and offers more long-term usefulness of the learned contents.
- No distinction between platform-independent and platform-specific concepts was provided. One of the consequences is that, at the end of the course, students tend to believe that they only knew how to develop distributed systems with JEE (and not, for instance, with .NET or other platforms). Their perception was that all acquired knowledge was specifically for that platform.

In addition to this list of problems, there were a couple of external driving forces that helped us realize that a complete reorganization of the course goals and structure was required:

- The creation of the European Higher Education Area (*Bolognia process* [5]) to unify the different higher education systems of the European member states force all courses to be described in terms of the list of competences students will achieve after completing the course.
- Several standards and software development trends have appeared, matured and/or been popularized since the contents of the original course were defined, not only at the modeling level (MDD, MDA, RM-ODP) but also at the technological level (web services, EJB 3.0, web applications frameworks, etc.). We thought that these new

concepts were, at least, worth mentioning and also related to the existing contents.

5. EDUCATIONAL OBJECTIVES

Like any other engineer, software engineers must learn, for any discipline, its theoretical foundations, its design methods, its standards, and its technological solutions and tools. Moreover, they must also master the conceptual issues that discriminate good solutions from merely valid ones.

Currently, most distributed software developers are educated mainly in the specific tools and technologies for writing and managing distributed software systems. Given the continuously and rapidly evolving nature of software, a common problem is to decide how a very broad range of technology is embraced.

Instead, we believe that education on DSE should especially emphasize its underlying principles. Students should be equipped with skills that allow them to understand and manage the various technological trends without being overwhelmed by their complexity and fast evolution, preventing them from the danger of rapid obsolescence. This should be accomplished without ignoring the classical development skills, and ensuring the cohesion and effectiveness of mental models and tools.

To address these issues, we decided to rethink the DSE course according to the following set of general educational goals:

- emphasize the stable and long-lasting concepts of this discipline;
- use, when possible, well-established models and notations; international standards are key to any engineering discipline, in particular to SE [6];
- focus on how to select and evaluate different methods and approaches rather than follow them as recipes;
- insist on a critical and comparative attitude;

More specifically, our goal is for our students to acquire the following skills. By the end of the course, they should:

- be aware of the different concerns and aspects that need to be considered when developing distributed applications;
- identify the different software architectural styles and be able to define the most suitable software architecture according to the particular characteristics of each application;
- be conscious of the similarities and differences between the different technological platforms currently available;
- realize that the same software development process can be used independently of the final implementation platform, since all of them rely on the same architectonic principles;
- understand how component-oriented programming can serve as a useful implementation technique for software systems;
- know how to transform the (platform-independent) specification of a distributed software system into a software design for current distributed and component-based technological platforms, with special attention to JEE, and
- be able to fully implement a distributed system (for the particular case of the JEE platform). This is important so that students see all phases of the software development process and the relationships between the artifacts obtained in each phase.

In the following section, we will discuss how we reshaped the DSE course in order for our students to acquire all these skills and knowledge.

6. THE NEW COURSE DESIGN

After stating our educational priorities, we decided upon the new course contents (section 6.1) and organization (6.2) in order to help students in achieving them.

6.1 Course Contents

The course is intended to provide an introduction to the concepts and fundamental methods for the design and development of component-based distributed systems, thus complementing the knowledge acquired in previous courses. In this way, the course explains both the theoretical concepts in the design and development of distributed systems, and the way in which the present technological platforms implement such concepts.

Our main goal was to combine an eminently practical approach with a solid conceptual framework, independent of the implementation technology, so that the problems described in Section 4 can be overcome. Moreover, the use of a conceptual framework allows us to present all contents in a more coherent way and easily relate the different phases of the software development process.

In addition, we also tried to make use of international standards whenever they were available, since they need to play a cornerstone role in SE (as they do in any other mature engineering discipline).

We finally decided to structure the new contents of our DSE course into five different modules, which are described in the following sections.

6.1.1 Module 1: viewpoints for distributed systems

Normally, the different aspects to be considered in a distributed application are intermingled, thus increasing the complexity of its design and implementation. Even worse, it is not very common to apply any systematic method to handle these concerns separately, nor to integrate them in a controlled way. Thus, handling them is usually rather chaotic, and dependent on the programmer or designer at hand. The Software Engineering community has therefore come to the conclusion that these problems must be addressed in the initial stages of the software development process. Specifically, at the architectural level, when decisions on the structure, general goals and strategies, implementations platforms and system deployment are taken.

Consequently, most of the existing proposals for describing the global architecture of distributed systems (i.e., the *enterprise architecture*) are based on the identification and separation of independent viewpoints, as prescribed by IEEE Std. 1471. Each one of these viewpoints focuses on a single aspect of the system, abstracting from the rest, and thus simplifying the design. In fact, several standards have been published (e.g., Kruchten's "4+1" views model, the Zachman framework, DoDAF, TOGAF, FEAF,

or the RM-ODP), which try to settle the basis for the description of the global architecture of software systems, using this modular separation of the design in different perspectives. We believe that the use of international standards is the most effective way to achieve the required interoperability between the different parties and organizations involved in the design and development of complex systems.

Thus, among all the available standards and proposals, we have chosen RM-ODP (Reference Model for Open Distributed Processing [7]) as the framework to be used in our DSE course. The rest of the standards mentioned above are presented briefly. RM-ODP is a joint ISO/IEC and ITU-T standard which is currently receiving increasing interest from many large companies and organizations, such as NASA/JPL, INTAP, EDF, etc. It provides a comprehensive and coherent framework of concepts for the specification of complex large scale IT systems, and now it has taken on new significance in the light of the MDA initiative from the OMG and the wide-scale adoption of Service-Oriented Architectures (SOA). In addition, major companies and organizations are starting to use RM-ODP as an effective approach for structuring their large-scale distributed IT system specifications, mainly because the size and complexity of current IT systems is challenging most of the current software engineering methods and tools. These methods and tools were not conceived for use with large, open and distributed systems, which are precisely the systems that the RM-ODP addresses.

RM-ODP defines five different and complementary viewpoints: enterprise, information, computation, engineering, and technology. Each of these viewpoints is studied separately in the course. Besides, in RM-ODP, an "abstract" language is defined for each of the five viewpoints. They are abstract in the sense that they define what concepts should be used but not how they should be represented. However, several notations have been proposed [8], although we have opted for the general purpose modeling notation UML. UML is familiar to our students, is easy to learn and to use by non-technical people, offers a close mapping to implementations, and has commercial tool support. Furthermore, the use of UML for ODP system specification has recently been standardized by ISO/IEC and ITU-T [9], which allows us again to align our course contents with international standards—something which is particularly important in any engineering discipline.

In summary, this first module serves as an introduction to the concepts and mechanisms on which RM-ODP bases the architectural description of distributed systems using independent viewpoints. In addition, it reviews the main international standards related to these subjects, which guarantee portability, interoperability and compatibility between applications developed by different enterprises or organizations.

From the five ODP viewpoints, in this course we concentrate on three of them: the computational viewpoint, which provides the high-level description of the software architecture and functionality of the system in a platform and technology independent manner; the engineering viewpoint, which describes the concepts and mechanisms used for the distribution of that functionality across different physical nodes, i.e., machines and processes; and the technology viewpoint, which describes how the ODP system is implemented. Figure 2 clarifies these five viewpoints and draws the mapping to the course modules.

6.1.2 Module 2: architectural styles for the development of distributed systems

The computational viewpoint of a distributed system determines its software architecture by means of a high level description of its functionality in terms of architectural components, interfaces and connectors. Thus, the architectural components encapsulate the basic system's functionality, provided through the component interfaces, whereas the connectors describe how these interfaces are related to achieve this functionality. In this module, students become familiar with the advantages of having an independent technology design for this viewpoint, i.e., they learn a design that is not focused on how the architectural components will be later distributed in physical nodes or implemented.

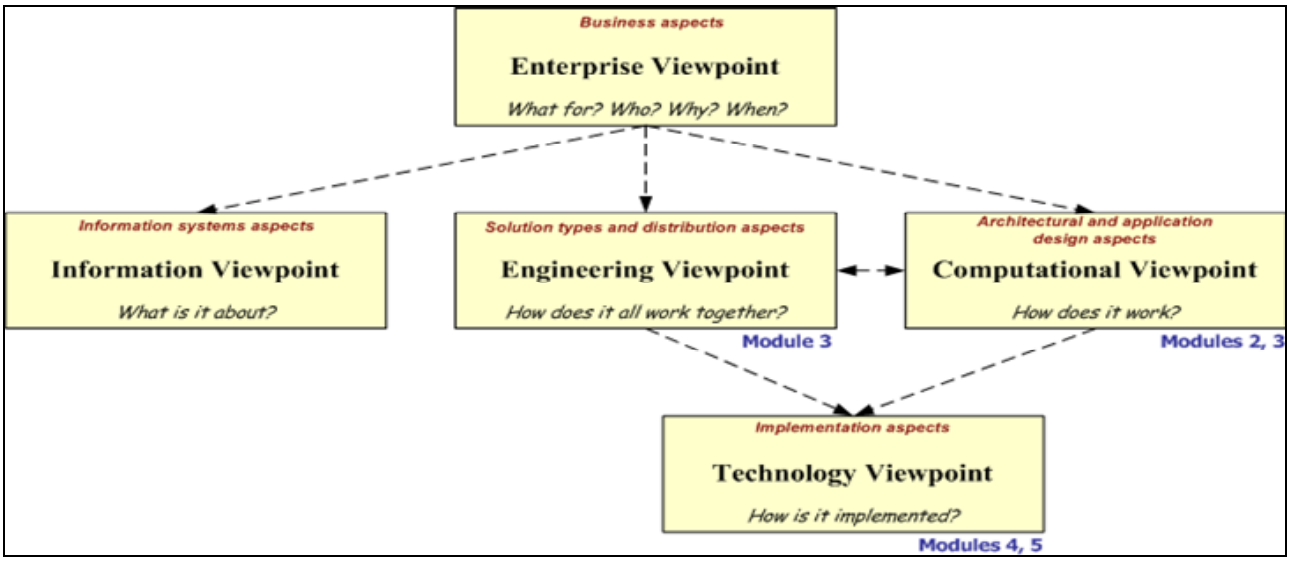

Figure 2. Overall view of the proposed framework and its application over an example model.

In this regard, we place special emphasis on the importance that the correct choice of an architectural style has in the development process. We present the student with commonly used architectural styles for the development of distributed systems (e.g., multi layers, client-server and peer-to-peer architectures) though we focus on the three-layer (or n-layer) architecture due to its importance in the development of web applications.

At the end of this module, students are able to suggest (among a predefined catalogue of typical architectural styles) and adapt an architectural style to meet a concrete system specification, taking into account both its functional and non-functional requirements (such as performance, scalability, etc). This is not an easy task, since we may have several appropriate architectures that suit the system specification. Thus, to decide which is the best one requires compromising a set of quality criteria (e.g., simplicity, extensibility, performance, etc.), many of which are usually contradictory to each other.

In addition, this second module tries to show the student how to describe the architectural design using UML. This aim also requires that the student understands, on the one hand, the role of the architectural design and its relevance in the development process of distributed systems and, on the other hand, the importance of reusing existing architectural solutions to address a new system design with similar characteristics.

6.1.3 Module 3: component-oriented programming as a technique for the realization of software architectures

Once we have selected a specific architectural pattern for our application, and the description of the software architecture has been made, we need to develop the system. It is possible to use different alternatives, depending on the programming paradigm chosen: structured programming, object-oriented programming, component-oriented programming, aspect-oriented programming, etc. Each of these paradigms comes with different technologies and programming languages, suitable for specific types of systems. This module focuses on one of these paradigms, namely component-oriented programming (COP), currently the most widely acknowledged and used approach for developing distributed applications.

It is important to note that the COP concepts, mechanisms and processes presented in this module are described in a general way, independently of any specific platform or implementation technology, in order to separate the concepts particular to this discipline from their implementation in any concrete platform (commercial technologies evolve much faster than their supporting theoretical concepts). Hence, in this module we adopted the definition of "component" (proposed in [10]) that considers that "the specification of a component represents the specification of a software unit and describes both the provided services, as well as the required ones from other components, and the behavior of any component instance concerning to its specification." These "component specifications" define the abstract components that comprise the system design and refine the previous ones identified in the computational viewpoint specification although there may not exist a one-to-one correspondence among them. That is, an architectural component may be implemented by several different interconnected software components, which jointly realize the services offered by the specification of such component.

RM-ODP does not prescribe any specific development process to define the tasks that must be performed by the software engineer to "transform" the architectural components and connectors into software components. Note that at the specification level we refer to architectural components instead of referring to software components. The latter implement the functionality of the architectural components defined by the software architecture of a system (i.e., software components realize architectural components). Thus, in this course we offer the students a generic vision of the different software development processes that allow us to implement the requirements of the software architecture from these software components. In any case, and for practical reasons, we try to follow the Cheesman and Daniels approach [10], because it is intrinsically easy and because it is well-known and widely adopted in practice.

Architectural and software components are described by means of UML 2 component diagrams. All other concepts of the RM-ODP engineering viewpoint (those related to distribution, remote access, etc.) as well as other physical structures (as DLLs, executable files, etc.) are described and represented with UML deployment diagrams.

6.1.4 Module 4: implementation with JEE

In this module, the student gets familiar with how the theoretical concepts studied in the previous modules are implemented in a specific technology platform (technology viewpoint)

In particular, this module explains in detail the principles of the JEE platform, its elements and the architectural patterns it provides. The module is focused on the study of multi-layer architectures, as proposed by the JEE application model, and analyzes the components and technologies offered by this particular platform in each layer (the Enterprise Java Beans, the Java Server Pages, etc.).

Once the student has acquired a basic knowledge of the JEE platform, he/she learns how to transform the platform-independent specification of the distributed software system (obtained as a result of defining the RM-ODP viewpoints described above) into a platform-specific design for the JEE platform. If not before, during this translation the student finally realizes that the contents of the first three modules can be used regardless of the technology platform where the system is about to be implemented.

The last part of this module provides some recommendations for helping to choose the right JEE technology for each part of the system during the translation from the RM-ODP specification to the JEE design. These recommendations are mostly based on the professional experience of the course consultants.

6.1.5 Module 5: other technological platforms

Finally, this last module introduces alternative platforms to JEE, which also provide valid implementations of the technology viewpoint of RM-ODP. This module is also intended to provide a historical view of the implementation technologies that can be used for the development of distributed applications. The main

principles of CORBA, Microsoft .NET and the Web Services implementation platforms are explained and illustrated here using the same simple distributed application. Thus, the skills and knowledge acquired by the student in this module will allow him/her to establish the main similarities and differences between the different component frameworks available on the market for implementing distributed applications.

Additionally, we believe this comparison helps students become familiar with the main keywords and acronyms used by the different platforms, and also improves the student's confidence in his/her abilities to develop distributed systems in any kind of technology platform (because they discover that these technologies are just implementations of the basic concepts and mechanisms that they have learnt previously, mainly using different terminology).

6.2 Course Organization and Assessment

This course lasts for an entire term (14-15 weeks) and includes four assessment activities. The activities cover the different phases of the software development process for a proposed distributed software system (from analysis to implementation). In what follows we describe the course scheduling, indicating for each week the modules to study, the activities to perform and typical tasks requested in them.

- Weeks 1-2. Description of the course planning, study of module 1, and a first look at module 2.
- Weeks 3-4. First assessment activity. As main tasks, it includes the definition of the information viewpoint (i.e. the domain model system, specified using the knowledge acquired in the previous SE courses) and an initial software architectural description (module 2) for the proposed software system. Additionally, the student must comment on the architectural styles of a set of real systems listed in the activity.
- Weeks 5. Study of module 2.
- Week 6-7. Second assessment activity. Starting from a given solution of the information viewpoint provided by the consultant, in this second activity the students must define the computational viewpoint, i.e., the complete software architecture of the system (module 2) and the dependencies between this viewpoint and the elements of the information viewpoint (i.e., how the functionality of the system handles and changes the system's persistent information).
- Week 8. Study of module 3 and first part of module 4.
- Week 9-10. Third assessment activity. Students must complete the specification of the system by defining its engineering viewpoint, i.e., how it is distributed across different nodes and computers (module 3) and then, transform this set of platform-independent models into a set of platform-specific design models for the JEE platform (module 4).
- Week 11. Complete the study of module 4.
- Week 12-15. Fourth assessment activity. Given the official design models for the example software system, the students must implement a JEE application that respects all design decisions expressed in all the different models (interfaces of each component, distribution, and so forth). There is an optional part that allows the students to propose (and implement) their own extensions to the basic system to account for further aspects and functionality.

Note that module 5 is not explicitly used during the practical part of the course. It was designed more as a complementary feature and to be used for student reference (even after finishing the course).

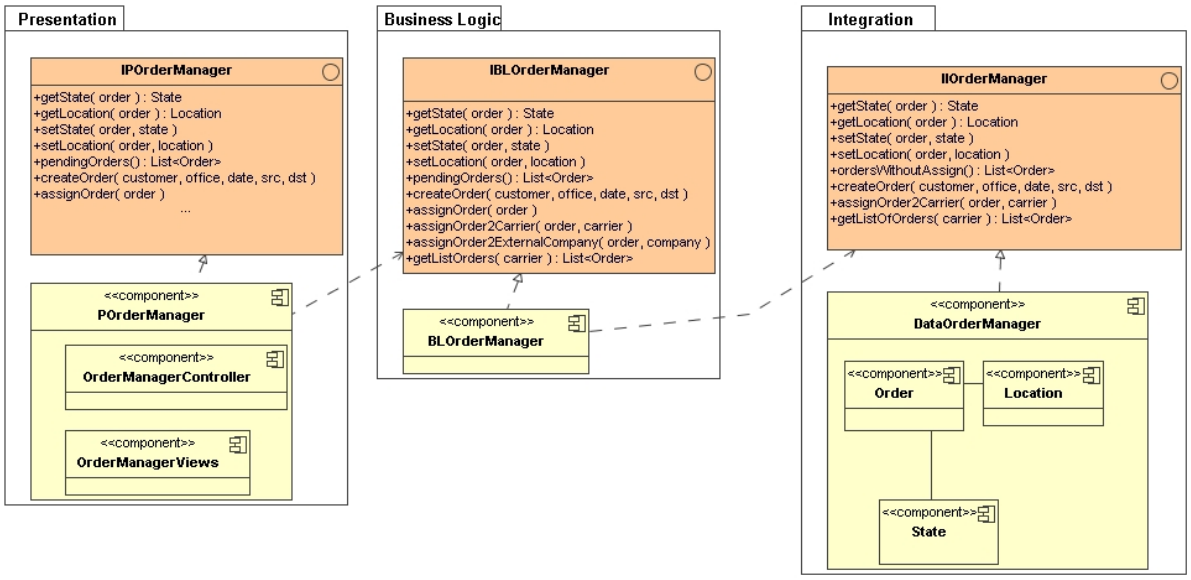

Figure 3. Example of the specification of a general software architecture for an order managements system (task included in the second assessment activity).

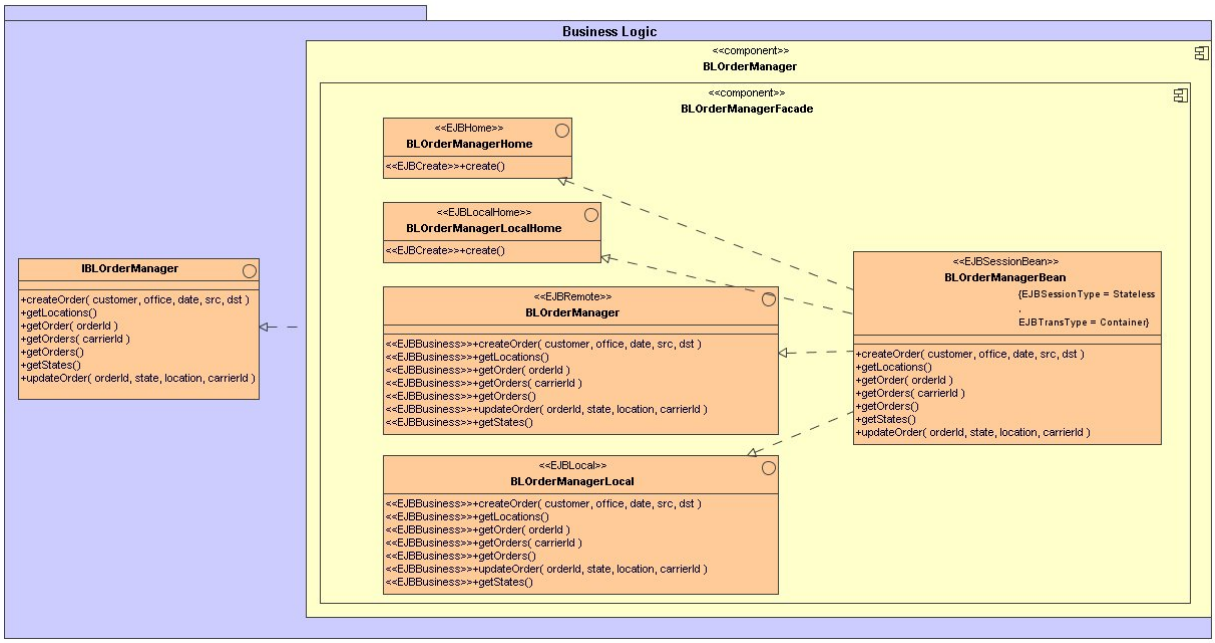

Figure 4. Example of the design of a subset of one of the previous architectural components for the JEE platform (task belonging to the fourth activity), after the refinement of the initial architectural components in terms of a set of software components (done as part of the third assessment activity; not shown here).

The final mark is obtained by combining the marks obtained in each individual activity (there is no final exam, quite unfeasible with virtual students.) Students that have not completed the fourth activity (i.e. who have not been able to provide a functional implementation of the system; one on the key skills we want our students acquire) automatically fail the course.

7. SOME REFLECTIONS AND PRELIMINARY CONCLUSIONS

After completing the first two terms with the new design of the DSE course, we are ready to present the first conclusions.

According to the formal polls sent to the students at the end of each term (these polls are anonymous and voluntary) the overall student satisfaction with the course has not changed. We believe that this is positive, because we were originally convinced that the satisfaction level would decrease: usually, students tend to prefer Note that, comparing both course organizations, in the previous course the students had ten weeks to develop the proposed JEE application programs, while in the new one they barely have four weeks to implement the same application. This is one of the most important trade-offs of the course. With four weeks time to learn to program in JEE they cannot master the details of this technology platform. However, we still believe it is important that students show they are at least able to develop a complete JEE application.

In this sense, and to help students grasp all the course concepts we have recently developed a complete case study (more than 100 pages), where a variant of the typical "online pet store" application (http://java.sun.com/developer/releases/petstore/) is specified in RM-ODP, designed for the JEE platform, and then completely implemented. Students can use this case study as a reference for all their assessment activities. We expect that this case study will help improve the student success rate in the future.

practical contents instead of more "abstract" conceptual courses. This is strengthen by the fact that, at least in Spain, the number of job offers containing the keyword JEE outnumbers by far those offers mentioning UML modeling or enterprise architecture, and thus, we thought that a shorter view of the JEE part in favor of the "UML" part would be hardly appreciated. Though not reflected in the formal poll, some students have also shown in informal communications their preference for this new approach. In particular, most students mention that it is now easier to understand the different aspects that must be considered in the design of a distributed system, and to understand the different concepts and mechanisms of commercial component platforms.

What worries us a little is that the student failure rate has increased by 10% (average). We believe that the main reason is that now the course includes much more content than before (all the "old" programming part plus the "new" modeling part).

Another issue revealed to be very important to ensure student success is providing technical assistance during the programming part of the course. Implementing an application with JEE requires installing and configuring an application server (JBoss in our case), an integrated development environment (Eclipse) and a database server (MySQL). To avoid students spending too much time with these low-level tasks, we provided a virtual lab for the course with a specific tutor who answers all installation and configuration questions within 24 hours.

From an organizational point of view, we would also like to mention that it is difficult to find consultants with a suitable profile to teach this course. Ideal candidates should possess both technical skills as well as in deep UML analysis and design capabilities. It is easy to find tutors that may cover part of the course but not to find tutors who are expert in the whole course contents. An alternative way to tackle this problem is to assign two different tutors to each virtual class, one in charge of the first three modules and the other focused on the last two.

8. ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their insightful and constructive comments and suggestions. This work has been partially supported by Spanish Research Projects TIN2005-09405-02-01 and TIN2005-06053.

9. REFERENCES

[1] Kramer, J. Distributed Software Engineering. ICSE'94, ACM Press, pp. 253-263, 1994.

[2] UOC. Open University of Catalonia. www.uoc.edu

[3] European Commission. ECTS - European Credit Transfer and Accumulation System. http://ec.europa.eu/education/programmes/socrates/ects/index_en.html

[4] Rodríguez, M. E., Serra, M., Cabot, J. and Guitart, I. Evolution of Teacher Roles and Figures in E-learning Environments. ICALT'06, pp. 512-514, 2006.

[5] Council of Europe. European Higher Education Area. http://www.coe.int/T/DG4/HigherEducation/EHEA2010/Default_en.asp

[6] Coalier, F. Standards, Agility, and Engineering. IEEE Computer, 40, 9, pp. 100-102, 2007.

[7] ISO/IEC. Open Distributed Processing - Reference Model: Foundations. ISO/IEC IS 10746 ITU-T Rec. X.901 to X.904, 1997. http://www.rm-odp.net

[8] Romero, J. R., Durán, F. and Vallecillo, A. Writing and Executing ODP Computational Viewpoint Specifications using Maude. Computer Standard & Interfaces, 29, 4, pp. 481-498, 2007.

[9] ISO/IEC. Use of UML for ODP system specifications. ISO/IEC FDIS 19793, ITU-T 2006. Rec. X.906, 2007. http://www.rm-odp.net

[10] Cheesman, J. and Daniels, J. UML Components. A simple process for specifying component-based software. Addison-Wesley, 2000.