Combining Generic Programming and Service-Oriented Architectures for the Effective and Timely Development of Complex e-Learning Systems

Santi Caballé

Open University of Catalonia, Department of Computer Science, Multimedia, and Telecommunication
Rambla. Poblenou, 156. 08018 Barcelona, Spain
scaballe@uoc.edu

Abstract

Over the last years, e-Learning needs have been evolving accordingly with more and more demanding pedagogical and technological requirements. On-line learning environments no longer depend on homogeneous groups, static content and resources, and single pedagogies, but high customization and flexibility are a must in this context. As a result, current educational organizations' needs involve extending and moving to highly customized learning and teaching forms in timely fashion, each incorporating its own pedagogical approach, each targeting a specific learning goal, and each incorporating its specific resources. Moreover, organizations' demands include a cost-effective integration of legacy and separated learning systems, from different institutions, departments and courses, which are implemented in different languages, supported by heterogeneous platforms and distributed everywhere, to name some of them. Therefore, e-Learning applications need to be developed in a way that overcome these demanding requirements as well as provide educational organizations with fast, flexible and effective solutions for the enhancement and improvement of the learning performance and outcomes. To this end, in this paper, an innovative engineering software technique is introduced that combines the Generic Programming paradigm and Service-Oriented Architectures in the form of Web-services for the effective and timely construction of flexible, scalable, interoperable and robust applications as key aspects to address the current demanding and changing requirements in software development in general and specifically in the e-Learning domain. This results in a generic, reusable, extensible platform called Collaborative Learning Purpose Library for the systematic development of collaborative learning applications that help meet these demanding requirements.

1. Introduction

Over the last decade, educational organizations' needs have been changing in accordance with ever more complex pedagogical models as well as with technological evolution resulting in e-Learning environments with very dynamic and changing teaching and learning requirements [1]. In particular, these needs involve extending and moving to highly customized learning and teaching forms in timely fashion, each incorporating its own pedagogical approach, each targeting a specific learning goal, and each incorporating its specific resources. Organizations' demands also include a cost-effective integration of legacy and separated learning systems, from different institutions, departments and courses, which are implemented in different languages, supported by heterogeneous platforms and distributed everywhere, to name some of them [1], [2].

In addition, collaborative learning environments [3], [4] must provide advanced enablement for distribution both of collaborative activities and of the necessary functionalities and learning resources to all participants, regardless the location of both participants and resources. The aim is to enable the collaborative learning experience in open, dynamic, large-scale and heterogeneous environments [5].

From this view, one of the main challenges in the development of modern e-Learning systems is to overcome important non-functional requirements arisen in distributed environments such as scalability, flexibility, availability, interoperability, and integration of different, heterogeneous, and legacy learning systems. Specific requirements include:

These requirements represent a great challenge for the latest trends of software development to be completely satisfied. To this end, software techniques and paradigms have been evolving all the time to mainly provide higher levels of abstraction so that developers can reuse and integrate not only functionality and components but more complex yet larger pieces of software. Moreover, although transparency has been greatly enhanced by current software techniques, the barrier of technology incompatibilities and the dependencies between components and clients make the transparency capability still difficult.

In this paper, an innovative approach of software engineering is presented which combines two emerging paradigms, namely generic programming [6] and service-oriented architectures [7]. These two views are later on merged by means of a recently well-known software development technique, namely Model-Driven Architecture [8]. This results in a generic, reusable, extensible platform called Collaborative Learning Purpose Library (CLPL) [9], which help overcome the demanding requirements appearing in the development of complex software in the collaborative learning domain.

The paper is organized as follows: Section 2 shows the background and related technologies involved in this approach. Section 3 describes the CLPL platform as a result of the described technologies while Section 4 shows the experience in the development of a complex collaborative learning application by using the CLPL. Section 5 ends the paper by summarizing the approach presented and drawing the main conclusions.

2. Background and related work

In this section, a brief overview of the existing technologies and paradigms related to this work is presented, namely Computer-Supported Collaborative Learning, Generic Programming, Service-Oriented Architecture, and Model-Driven Architecture. This overview will serve as background for the next sections.

2.1. Computer-Supported Collaborative Learning

Computer-Supported Collaborative Learning (CSCL) is one of the most influencing research paradigms dedicated to improve teaching and learning with the help of modern information and communication technology [3], [4]. Collaborative or group learning refers to instructional methods where students are encouraged to work together on learning tasks. As an example, project-based collaborative learning proves to be a very successful method to that end [4]. Therefore, CSCL applications aim to create virtual collaborative learning environments where students, teachers, tutors, etc., are able to cooperate with each other in order to accomplish a common learning goal.

To achieve this goal, CSCL applications provide support to three essential aspects of collaboration, namely coordination, collaboration and communication; with communication being the base for reaching coordination and collaboration [9]. Collaboration and communication might be *synchronous* or *asynchronous*. The former means cooperation at the same time and the shared resource will not typically have a lifespan beyond the sharing while the latter means cooperation at different times being the shared resource stored in a persistent support.

The representation and analysis of group activity interaction is an important issue in CSCL for the support of coaching and evaluation in online collaborative earning environments [3]. Interaction analysis relies on information captured from the actions performed by the participants during the collaborative process. To this end, fine-grained notifications and complex information collected from the learners' interaction are provided to give immediate feedback about others' activities and about the collaboration in general [10].

2.2. Generic Programming

In all advanced forms of engineering it can be observed that new products are usually developed by reusing tried and tested parts rather than developing them from scratch. The reuse of previously created product parts leads to reduced costs and improved productivity and quality to such an extent that industrial processes will take a great leap forward. Generic Programming (GP) [6], [11] has emerged over the last years to facilitate this possibility in the software engineering field.

GP is an innovative paradigm that attempts to make software as general as possible without losing efficiency. It achieves its goal by identifying interrelated high-level family from a common requirement set [6]. By the application of this technique, especially in design phases, software is developed offering a high degree of abstraction which is applicable to a wide range of situations and domains.

By applying GP to develop computer software important objectives are achieved:

- Reuse. This means to be able to reuse and extend software components widely so that it adapts to a great number of interrelated problems.
- Quality. Here ”quality” refers to the correctness and robustness of implementation which provides the required degree of reliability.
- Efficiency. It is also essential to guarantee the efficiency of components as if this not done the performance repercussions will be noted, just as with lack of quality, in all of the systems involved.
- Productivity. Inherent to reutilization is the saving through not having to create software components again that already exist. Hence, there is an increase in computing production.
- Automation. The aim is to automate the processes so that general requirements with a high level of abstraction and specially designed tools can be used to produce operative programmes.
- Personalisation. As the general requirements are made more particular, so the product that is generated becomes more optimised to meet the specific needs of the client.

GP also represents one important technique to achieve effective Product Lines (PL) following the Product-Line Architecture(PLA) approach [6]. PLA promotes developing large families of related software applications quickly and cheaply from reusable components. In PLA, a certain level of automation is provided in the form of generators (also known as component configuration tools) to realize solutions for large parts of the systems being developed.

2.3. Service-Oriented Architecture

Service-Oriented Architecture (SOA) [7] represents the next step in the software development to help organizations meet their ever more complex set of needs and challenges, especially in distributed systems [1], [5]. This is achieved by dynamically discovering and invoking the appropriate services to perform a request from heterogeneous environments, regardless of the details and differences of these environments. By making the service independent from the context, SOA provides software with important non-functional capabilities for distributed environments (such as scalability, heterogeneity and openness), and makes the integration processes much easier to achieve.

SOA relies on services. According to W3C [7], a service is a set of actions that form a coherent whole from the point of view of service providers and service requesters. In other words, services represent the behaviour provided by a provider and used by any requesters based only on the interface contract. Within SOA, services

- stress location transparency by allowing services to be implemented, replicated and moved to other machines without the requester’s knowledge,
- enable dynamic access as services are located, bound and invoked at runtime,
- promote interoperability making it possible for different organisations supported by heterogeneous hardware and software platforms to share and use the same services,
- facilitate integration of other existing systems and thus protect previous investments (e.g. legacy assets),
- rely on encapsulation as they are independent from other services and their context,
- enhance flexibility by allowing services to be replaced without causing repercussions on the underlying systems involved,
- foster composition from other finer-grained services.

Although SOA can be realised with other technologies, over the last few years Web services has come to play a major role in SOA due to lower costs of integration along with flexibility and simplification of configuration. According to W3C, a Web service is a software system identified by a URI, whose public interfaces are defined and described using XML [7]. Other systems may interact with the Web service in a manner prescribed by its definition, using XML-based messages conveyed by internet protocols.

The core structure of Web services is formed by a set of widely adopted protocols and standards [7], such as XML, SOAP, WSDL, and UDDI, which provide a suitable technology to implement the key requirements of SOA. This is so because these protocols allow a service to be platform - and language - independent, dynamically located and invoked, interoperable over different organization networks, and supported by large organisations (e.g., W3C consortium).

2.4. Model-Driven Architecture

The Model-Driven Development (MDD) paradigm and the framework supporting it, namely Model-Driven Architecture (MDA) [12] have been recently attracting a lot of attention given that it allows software developers and organizations to capture every important aspect of a software system through appropriate models [13]. MDA provides great advantages in terms of complete support to the whole cycle development, cost reduction, software quality, reusability, independence from the technology, integration with existing systems, scalability and robustness, flexible evolution of software and standardization, as it is supported by the Object Management Group (OMG).

In proposing MDA, two key ideas have had significant influence in OMG aiming at addressing the current challenges in software development [12]: service-oriented architectures (SOA) and product line architectures (PLA). As to the former, SOA provides great flexibility to system architectures by organizing the system as a collection of encapsulated services. Hence, SOA relies on services which represent the behavior provided by a component to be met and used by any other components based only on the interface contract. As to the latter, PLA promotes developing large families of related software applications quickly and cheaply from reusable components.

There are many views and opinions about what MDA is and is not. However, the OMG, as the most authoritative view, focuses MDA on a central vision [12], [13]: Allow developers to express applications independently of specific implementation platforms (such as a given programming language or middleware). To this end, OMG proposes the following principles for MDA developments: first, the development of a UML-based Platform Independent Model (PIM), second, one or several models which are Platform Specific Models (PSM). Finally, a certain degree of automation by means of descriptions is necessary for mapping from PIM to PSM.

3. A generic, service-oriented collaborative learning platform

The software engineering paradigms and techniques described in the last section are here merged and taken one step further by the development of a generic, robust, interoperable, reusable, component-based and service-oriented Collaborative Learning Purpose Library (CLPL) [5], [9].

The CLPL is based on the GP and SOA paradigms so as to enable a complete and effective reutilization of its generic components as a skeleton for the construction of any collaborative learning application by means of implementing the conceptualization of the fundamental needs existing in any collaborative learning experience.

The CLPL also provides full support to distribution, reusability, flexibility and interoperability as key aspects to address the current non-functional needs in software development in general, and specifically in the CSCL domain. To this end, Web-services are the implementation technology chosen for the CLPL given the widely adopted protocols and standards, which represents the very rationale of the this technology. These standards represent a suitable context to guarantee interoperability and scalability by taking great advantage of the distributed technologies.

In this section, first the main guidelines of the development of the CLPL are discussed and justified by means of the use of GP, SOA, and MDA principles. Then, the architecture of this platform is briefly described.

3.1. CLPL development

There a great deal of similarities between the pervasive and challenging collaborative learning needs and the benefits provided by SOA. As a result of this matching, SOA appears to be the best choice to support the development of the CLPL. Indeed, SOA enhances educational organizations by increasing the flexibility of their pedagogical strategies, which can be continuously adapted, adjusted, and personalized to each specific target learning group. Moreover, SOA facilitates the reutilisation of successful collaborative learning experiences and makes it possible for the collaborative learning participants to easily adapt and integrate their current best practices and existing well-known learning tools into new learning goals.

Over the last years, CSCL has become a complex and extensive domain. Therefore, the application of the GP principles appear to be a good choice for the development of the CLPL by, first, identifying those parts which are common to most applications of the CSCL domain. Then, proceed to isolate the fundamental parts in the form of abstractions from which the basic requirements are obtained. Finally, encourage the greatest possible reusability of the resulting generic components for the construction of as many CSCL applications as possible.

In order to turn the CLPL into an effective software platform, its development was based on the MDA approach. This paradigm fits very well in combination with the GP and SOA principles due to the clear separation of a generic, reusable technology-independent model from a different, flexible technology-dependent implementation models.

To this end, the first step was to create a PIM by applying the following Generic Programming ideas [11]: (i) define the semantics of the properties and domain concepts, (ii) extract and specify the common and variable properties and their dependencies in the form of abstractions found in the CSCL domain, and (iii) isolate the fundamental parts in the form of abstractions from which the basic requirements were obtained, analysed and designed as a traditional three-layer architecture (i.e. presentation, business and information). To this end, the PIM was expressed using UML as the standard modelling language promoted by the OMG.

The second step was to build two different PSM from the unique PIM achieved: A Java implementation in the form of a generic component-based library and a collection of WSDL files organized in directories that are automatically turned into generic web-services implemented in the desired programming language and allowing developers to implement the services according to specific needs. On the one hand, the Java programming language provides great predisposition to the adaptation and correct transmission of generic software design, which make the software highly reusable. It lacks of full interoperability between different programming languages though. On the other hand, in order to increase flexibility and interoperability, the SOA-based PSM provides great predisposition to be involved in distributed environments supporting different middleware and programming languages.

Finally, in order to automate as much as possible the transition from the PIM to the appropriate PSM, the latest research results are leading us to deal with XMI files (see [12] for details), which are XML-tagged files as the result of coding UML diagrams. In combination with XSL style sheets, it is possible to turn the PIM's XMI files into WSDL files, which represent the input for a Web-service working environment to transform them into a specific-language architecture design (PSM). Lack of comply with standard of the existing UML case tools is the major problem to face next as well as how to provide a more complete and detailed realization of the desired PSM.

The development of the CLPL fully followed the first and second steps while ongoing work is dealing with the last by introducing certain level of automation by means of WSDL descriptions.

3.2. CLPL architecture

The CLPL is mainly made up of five components which are independent according to its general internal functionality (see complete description in [9]):

CSCL User Management component: this contains all the logics related to the CSCL system user management which can act as a group coordinator, group member, group-entity and system administrator. It tackles both the basic user management functions in a learning environment and the user profile management. The latter implements the user and group models within a collaborative environment.

CSCL Security Management component: this contains all the generic descriptions of the measures and rules decided to carry out the authentication and authorization issues and so protecting the system from both the unknown users and the intentional or accidental bad use of its resources. Its genericity lets programmer implement them with the ultimate cryptographic security mechanism existing.

CSCL Administration Management component: this contains those specific data (through log files) and processes (statistical computations) so as to carry out all system's control and maintenance with the aim to administer the system correctly and to improve it in terms of performance and security.

CSCL Knowledge Management component: This supports a complete process of information and knowledge management. First, this component collects and classifies the system log files made up of all the events occurring in a certain workspace over a given period of time. Then, it performs the statistical analysis on the event information as well as the management and maintenance of the knowledge generated from this analysis.

CSCL Functionality component: this defines the three elemental parts involved in any form of cooperation, namely coordination, communication and collaboration [4]. Coordination involves the organization of groups to accomplish the important objectives of members such as workspace organization and group structure and planning. Collaboration lets group members share any kind of resources while communication represents the basis of the whole component since it enables coordination and collaboration to be achieved by providing them with low-level communication support. Furthermore, this component implements the presentation to users of the knowledge extracted by the previous component in terms of immediate awareness and constant feedback of what is going on in the system.

4. A CLPL application: a structured discussion forum

To illustrate the approach, a Web-based structured discussion forum called Discussion Forum (DF)[1] [5], [9] was developed to validate the possibilities offered by the CLPL. The aim was demonstrate both the

[1] The DF is found at http://einfnt2.uoc.edu:8090/df/login.php

effectiveness of developing a complex system in terms of quality, productivity and cost and the provision of new opportunities to learning methodologies, such as learning by discussion, that gives significant benefits to students in the context of project-based learning, and in education in general. This applications is currently running at the Open University of Catalonia[2] providing support to the discussion in several on-line courses.

In this section, first the pedagogical requirements of the DF are described and then some guidelines that conducted its design are provided. Finally, the implementation and deployment of the DF is reported.

4.1. Pedagogical background

In collaborative learning environments, the discussion process forms an important social task where participants can think about the activity being performed, collaborate with each other through the exchange of ideas that may arise, propose new resolution mechanisms, and justify and refine their own contributions and thus acquire new knowledge.

To this end, a complete discussion and reasoning process is proposed based on three types of generic contributions, namely specification, elaboration and consensus [9]. Specification occurs during the initial stage of the process carried out by the tutor or group coordinator who contributes by defining the group activity and its objectives (i.e. statement of the problem) and the way to structure the group activity in sub-activities. Elaboration refers to the contributions of participants (mostly students) in which a proposal, idea or plan to reach a solution is presented. The other participants can elaborate on this proposal through different types of participation such as questions, comments, explanations and agree/disagree statements. Finally, when a correct proposal of solution is achieved, the consensus contributions take part for its approval (this includes different consensus models such as voting); when a solution is accepted the discussion terminates.

4.2. The design of the application

The design of the DF includes certain thematic annotation cards (such as idea, evaluation, reply that structure the elaboration phase and can offer full help support as well. All events generated are to be recorded as user actions, which are then analyzed and presented as information to participants either in real time (to guide directly students during the learning activity) or after the task is over (in order to understand the collaborative process). To that end, the *CSCL Knowledge Management* and *CSCL Functionality* components provided full support to the event management. In particular, during the elaboration phase, a complete treatment of the structured task performance events generated enables the system to keep participants aware of the contributing behavior of others, to check certain argumentative structures during discussion and also to open up the possibility to provide feedback based on the data produced. Equally, group analysis outcomes produced by the treatment of group functioning events constitute an important data source that can assist in achieving a more satisfactory solution to the problem during the consensus phase. Furthermore, the coordinator can use this same information to organize well-balanced groups during the specification phase.

Personal features of the discussion group participants (their role, collaboration preferences and so on) were taken into account and a user and group model were designed so as to allow participants to add new services whilst their needs evolve as the discussion moves forward. All these user features were included by the *CSCL User Management* component through the user profile management subsystem, providing a solid support for building and maintaining the user and group model.

Therefore, the DF constitutes a valuable learning resource that takes advantage of the CLPL to greatly improve essential features of a discussion process such as awareness of participant contributions.

4.3. Implementation and deployment issues

By taking great advantage of the service-oriented approach of the CLPL functionalities, the primary principle was in the DF implementation was to provide a broad set of independent fine-grained Web-services grouped by a particular purpose [5], such as the authentication process and the presentation of the feedback extracted [10].

A clear, independent, and separated vision of each single behavior of the DF into fine-grained task-specific web-services resulted in a natural distribution of the application into different nodes in a network. This distribution was driven by matching the web-service purposes and the node configuration and location in the network. According to this view [5], the

[2] The Open University of Catalonia (UOC) is located in Barcelona, Spain. The UOC offers distance education through the Internet to 35,000 students

web services in the user interface layer should be allocated nearby the client; the business web-services would be better suited if allocated in those nodes with high-performance processors, and, finally, the data web-services could be attached or nearby the database supported by nodes with high storage capability. As to the database, it can be also distributed as it is clearly separated from the data web-services, which would be in charge of updating and keeping the consistency of the different instances of the database.

The ultimate goal was to enhance and improve the effectiveness of the learning experience in terms of non-functional requirements, such as robustness, scalability, interoperability and so on. Indeed, by installing and deploying replicas of the web-services all over the network fault-tolerance is easily achieved by redirecting a request to an exact replica of the web-service when a node is down. Concurrency and scalability become natural in this context by parallelizing the users' requests using as many replicas as necessary. Finally, interoperability is inherent in the context of web-services technology as they are fully independent from hardware platforms and programming languages.

In overall, the reuse of the CLPL's components in all phases of the DF development provided a severe reduction of cost in terms of time and effort while keeping quality and robustness high. This allowed the provision and later extensions of a complex application to support collaborative learning in timely fashion.

5. Conclusions

This paper proposes a step further in the current software development methodologies by taking advantage of the most advance and latest techniques in software engineering, such as Generic Programming and service-oriented architectures. The goal is to greatly improve software development in terms of quality, productivity and cost, as well to provide effective solutions to meet demanding non-functional requirements. To this end, an architectural solution in the form of a generic, service-oriented computational model called CLPL is provided to help develop complex, modern and advanced CSCL applications. Both the development experience of the CLPL and of a specific application based on this platform is reported to validate the key ideas proposed in this paper.

Ongoing work is to automatically describe WSDL files from the PIM model so that it is possible to generate PSM implementations of the CLPL in different programming languages and middleware.

Acknowledgments

This work has been partially supported by the Spanish MCYT project TSI2005-08225-C07-05.

6. REFERENCES

1. Pankatrius, V., Vossen, G., Towards E-Learning Grids: Using Grid Computing in Electronic Learning. Proceedings of IEEE Workshop on Knowledge Grid and Grid Intelligence, Halifax, New Scotia, Canada, (pp. 4-15), 2003.
2. Zaheer Abbas, Muhammad Umer, Mohammed Odeh, Richard McClatchey, Arshad Ali, Farooq Ahmad, A Semantic Grid-based E-Learning Framework (SELF). Proceedings of CCGrid 2005, Cardiff, UK, 2005.
3. Koschmann, T., Paradigm shifts and instructional technology. In T. Koschmann (Ed.), CSCL: Theory and Practice of an Emerging Paradigm, Mahwah, New Jersey, Lawrence Erlbaum Associates, (1-23), 1996.
4. Dillenbourg, P, Introduction; What do you mean by "Collaborative Learning"? P. Dillenbourg (Ed.), Collaborative learning. Cognitive and computational approaches, 1-19. Oxford: Elsevier Science, 1999.
5. Caballé, S., Xhafa, F., Daradoumis T. A Service-oriented Platform for the Enhancement and Effectiveness of the Collaborative Learning Process in Distributed Environments. Proceedings of the GADA 2007. Algarve, Portugal. LNCS, 2007.
6. K. Czarnecki, & UW Eisenecker (2000). Generative Programming: Methods, Techniques, and Applications. Boston, MA: Addison-Wesley.
7. Web Services Architecture Document. W3C Working Group, 2004 http://www.w3.org/TR/ws-arch/ (Web page as of November 2007).
8. Object Management Group: Model-Driven Architecture http://www.omg.com/mda (web page as of November 2007).
9. Caballé, S., Daradoumis, Th., Xhafa, F. (2007). A Generic Platform for the Systematic Construction of Knowledge-based Collaborative Learning Applications. Architecture Solutions for e-Learning Systems. Idea Group Press.
10. Zumbach, J., Hillers, A., & Reimann, P. (2003). Supporting Distributed Problem-Based Learning: The Use of Feedback in Online Learning. T. Roberts (Ed.), Online Collaborative Learning: Theory and Practice (pp. 86-103), Hershey, PA: Idea Group Press.
11. Caballé, S., & Xhafa, F. (2003). A Study into the Feasibility of Generic Programming for the Construction of Complex Software. In: Proceedings of the 5th GPCE/Net.Objectsdays 2003.
12. Object Management Group: Model-Driven Architecture http://www.omg.com/mda (web page as of November 2007).
13. K. Czarnecki. Overview of Generative Software Development (2005). In J.-P. Banâtre et al. (Ed.), Unconventional Programming Paradigms (UPP) 2004, Lecture Notes in Computer Science: Vol. 3566 (pp. 313-328). Berlin: Springer-Verlag.