UNIVERSIDAD DE VALLADOLID

Dpto. de Teoría de la Señal, Comunicaciones e Ingeniería Telemática

Escuela Técnica Superior de Ingenieros de Telecomunicación

Tesis Doctoral

GLUE!: An Architecture for the Integration of External Tools in Virtual Learning Environments

AUTOR

Carlos Alario Hoyos

Ingeniero de Telecomunicación

DIRECTORES

Miguel L. Bote Lorenzo

Dr. Ingeniero de Telecomunicación

Eduardo Gómez Sánchez

Dr. Ingeniero de Telecomunicación

Junio de 2012

A mis padres.

Acknowledgements

The opening lines of this document serve to close this dissertation. In them, I want to sincerely thank all those who have made this work possible, and without whom this dissertation could not have been accomplished. First, I would like to thank my supervisors Miguel and Eduardo, with whom I share the credit of this work, and who have supported me throughout all these years in which we have worked together very closely. They believed in me from the very first moment I ran into them, and they spent much time and effort to make this dissertation succeed.

Yannis is also one of the great minds that has most influenced me over these years, and so I owe him much of this work. He insisted in showing me the right way, and he never hesitated to give me on-the-fly advice or answers to my questions, despite his busy schedule. Of course, I have to specially thank Asen, Guillermo and Adolfo, who have very closely followed my dissertation, and with whom I had deep and intense discussions.

Regarding the development tasks, I have been very lucky to be supported by David and Javier; without their work, this dissertation would have needed much more time, and the developed systems would have been simpler and poorer. Besides, I learnt a lot from all the people working in the GSIC-EMIC research group; people like Chus, Luis Pablo, Iván, Eloy, Osmel, Juan or Alejandra among many others, with whom I shared the good and the bad days in the laboratory. They all made the working environment a dynamic and pleasant place to spend the long working hours.

I would also like to use these lines to thank the hospitality of the people I visited in the research stays I made during my PhD. First, thanks to those working in the Institute for Educational Cybernetics in the University of Bolton, with special thoughts for Scott Wilson, Bill Olivier, Dai Griffiths and Mark Johnson. Also, I would like to include in these acknowledgements the people I visited in the different Universities in the Sydney area, highlighting James Dalziel, who welcomed me at the MELCOE, Peter Goodyear and Sue Bennett.

Finances are normally a terrible headache on research. Nevertheless, I was very fortunate to participate in several European projects (IST-FP6-507838, IST-FP6-034567), national projects (TIN2008-03023, TIN2011-28308-C03-02, IPT-430000-2010-054), and regional projects (VA293A11-2, VA301B11-2), in which the GSIC-EMIC was involved, and that somehow served to partially cover my stipend and research expenses at different stages of my PhD. It is also noteworthy the fellowship I received from the Training of Research Staff (FPI) Spanish program (BES-2009-014397), associated to one of the aforementioned national projects (TIN2008-03023). This fellowship gave me economic stability for almost three years and allowed me to make several short research stays abroad.

Despite this ideal working environment, I could never have finished this dissertation without the support of my family and friends. Among them, special words of thanks are to Martín,

Eduardo, Natalia, Miguel Ángel, Dani, Quini and many others who stood by my side to a greater or lesser extend throughout these years. Here, I would also like to remember all my college friends which led me to the front door of this PhD after a five-year degree in which we all shared great times and memories.

My parents, Julián and Tere, deserve my best words. They raised me in the values of sacrifice and knowledge. They always backed me up in every choice, and helped me overcome the difficulties, and what is more important, they still keep doing it day by day. They are the ones that encourage me to get the best out of myself.

Finally, my last words of gratitude must be for Bea, because she suffered this dissertation as I did. She was always by my side and never gave up on me. She gave me the strength to face this challenge, being even able to follow me to the end of the world. You have been the most important support for me these years. I have no other words, but THANKS!

Abstract

The integration of external tools in Virtual Learning Environments (VLEs) aims at enriching the learning activities that educational practitioners may design and enact. Traditionally, external tools have been integrated by means of *ad hoc* developments, being this solution very inefficient as the number of VLEs and tools employed by practitioners increases. Besides, those generic approaches tackling the integration of multiple external tools in multiple VLEs have not found a broad adoption, mainly because of the high development effort required to integrate new tools and VLEs, and the restrictions imposed on VLE and tool providers. Some recent works tried to overcome these two limitations by proposing a lightweight integration of tools. Nevertheless, these works do not facilitate the instantiation and enactment of collaborative learning situations, which significantly precludes practitioners from employing VLEs distinctive collaborative features for the management of users and groups.

This dissertation proposes a middleware integration architecture called GLUE! (Group Learning Uniform Environment) that enables the lightweight integration of multiple existing external tools in multiple existing VLEs, overcoming these limitations. GLUE! fosters this integration by imposing few restrictions on VLE and tool providers, as well as by expecting an attainable effort from developers. Besides, GLUE! facilitates the instantiation and enactment of collaborative learning situations within VLEs, leveraging the VLEs distinctive features for the management of users and groups. By means of GLUE!, practitioners may use external tools as if they were VLE built-in tools, and without having to give up the VLEs they are used to.

GLUE! has been evaluated using three authentic collaborative learning situations that were designed to meet the pedagogical needs of three different higher education courses. These three situations were employed in four different experiments involving real educators and students. The results of this evaluation show that GLUE! allows the instantiation and enactment of collaborative learning situations that require the integration of external tools, reduces the burden associated to the instantiation of complex collaborative activities, and facilitates students the realization of these activities in collaboration. Interestingly, the development effort required by the proposed integration software was similar to that in other lightweight generic approaches that offer a lower degree of functionality.

Resumen

La integración de herramientas externas en VLE (*Virtual Learning Environments* - Entornos de Aprendizaje Virtual) tiene como objetivo enriquecer las actividades de aprendizaje que los profesionales de la educación pueden diseñar y poner en marcha. Tradicionalmente, las herramientas externas han sido integradas mediante desarrollos *ad hoc*, siendo esta solución muy poco eficiente a medida que aumentaba el número de VLE y herramientas utilizados por estos profesionales. Además, aquellas aproximaciones genéricas que abordan la integración de múltiples herramientas en múltiples VLE no han conseguido obtener una amplia adopción, principalmente debido al alto esfuerzo de desarrollo necesario para integrar nuevas herramientas y VLE, y a las restricciones impuestas sobre los proveedores. Algunos trabajos recientes han intentado superar estas dos limitaciones proponiendo una integración ligera de herramientas. Sin embargo, estos trabajos no facilitan la instanciación y puesta en marcha de situaciones de aprendizaje colaborativo, lo que impide de forma significativa que se puedan emplear las propiedades colaborativas específicas que proporcionan los VLE para la gestión de usuarios y grupos.

Esta tesis propone una arquitectura *middleware* de integración denominada GLUE! (*Group Learning Uniform Environment* - Entorno Uniforme de Aprendizaje en Grupo) que permite la integración ligera de múltiples herramientas externas existentes en múltiples VLE existentes, superando estas limitaciones. GLUE! fomenta esta integración imponiendo pocas restricciones sobre los proveedores de VLE y herramientas, así como demandando un esfuerzo asumible por parte de los desarrolladores. Además, GLUE! facilita la instanciación y puesta en marcha de situaciones de aprendizaje colaborativo desde los VLE, aprovechando las propiedades específicas de éstos para la gestión de usuarios y grupos. Por medio de GLUE!, los profesionales de la educación pueden utilizar herramientas externas como si fueran herramientas nativas de los VLE, y además sin tener que renunciar a los VLE a los que están acostumbrados.

GLUE! ha sido evaluado con la ayuda de tres situaciones de aprendizaje colaborativo auténticas, las cuales fueron diseñadas para cubrir las necesidades pedagógicas de tres cursos de educación superior. Estas tres situaciones se utilizaron en cuatro experimentos diferentes con educadores y estudiantes reales. Los resultados de esta evaluación mostraron que GLUE! permite la instanciación y puesta en marcha de situaciones de aprendizaje colaborativo que requieran la integración de herramientas externas, reduce la carga asociada a la instanciación de actividades colaborativas complejas, y facilita a los estudiantes la realización de estas actividades en colaboración. Curiosamente, el esfuerzo de desarrollo necesario por el software de integración fue similar al de otras aproximaciones de integración genéricas que ofrecen un menor grado de funcionalidad.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The use of Information and Communication Technologies (ICTs) in many different domains like medicine, industry, or education [Abb01] is changing society in a way that some years ago would have been difficult to foresee [Duq05]. The quick rise of Internet, web technologies, wireless networks and mobile devices have led to the adoption of ICTs not only at work or at school, but also in our daily lives, in a very short period of time [Pel01]. This offers the opportunity of ubiquitous communication among people around the world, as well as ubiquitous access to many services and data [Tho08].

Education takes advantage of these trends regarding ICTs in a field of study called Technology-Enhanced Learning (TEL) [Joh04], which has been under research for more than two decades [Gul08]. TEL studies the support of learning activities through technology at different levels, from primary school to higher education [Bat03], and at different contexts, including formal and informal learning [Era04, Fol06], but considering the challenges involved in the lifelong learning process [Sha00]. Research on TEL also includes distance learning (sometimes referred as e-learning) [Ros11], face-to-face learning and blended learning [Osg03], which combines traditional face-to-face and online activities. The important community of researchers working on TEL can be exemplified through outstanding active projects, such as STELLAR[1] (with more than fifteen research partners working on this project all over Europe), specific calls on this topic in several international funded programmes, like The Seventh Framework Programme[2], as well as through numerous specific journals, conferences, and recent publications on this field [Bal09, Hak08].

Within TEL, Computer Supported Collaborative Learning (CSCL) [Kos96] constitutes a multidisciplinary paradigm in which ICTs are used to facilitate the social and effective interactions among participants in the acquisition of knowledge and skills [Dil99, Mat97]. Practitioners

[1] `http://stellarnet.eu`. Last visited: June 2012.

[2] `http://cordis.europa.eu/fp7/home_en.html`. Last visited: June 2012.

with different backgrounds on psychology, education or technology generally participate in the development of CSCL systems, which are software applications that support the design, instantiation and enactment of collaborative learning situations [Her06b, Kop05]. In the same way that TEL, CSCL also presents an important community of researchers, as it can be seen in remarkable active projects such as Euro-CAT-CSCL[3], an international project with partners from three different countries funded by the European Commission, as well as in numerous journals, conferences, and recent publications on this topic [Bud08, Dil09].

According to [Dil99], collaborative learning situations are those that lead to the achievement of learning through collaboration. Authors in [Osu99] analyze five features that describe collaborative learning situations. The first feature is the *social configuration*, which comprises the description of the participants and their roles (student, educator, administrator, instructional designer, etc.), as well as the group structure [Mar04]. The *learning objectives* express both the individual and group goals that the participants must achieve during the enactment of the situation; these goals should be formalized in the design of the collaborative learning situation [Dil02a]. The situation has a *structure*, which can be decomposed in a set of activities [Gif99], each oriented to the accomplishment of a set of tasks [Dil02a]; this structure can sometimes be organized following a certain order of activities, thus forming a sequence of activities [Dal03]. Here, it is important to point out that collaborative learning situations may include both individual and collaborative learning activities [Osu99, Sta06]. In structured collaborative learning activities, participants can build strong relationships in order to achieve a common goal; as opposed to unstructured collaborative learning activities, in which participants do not share goals, and a minimum dependency among them is required [Chi02]. Another feature that should be defined is the set of *resources* supporting each learning activity [Her06a]; these resources may be tools, resources or other artifacts [Pal08]. Finally, the situation happens in an *environment*, in which participants find the structure of activities, tools and resources, and in which the objectives and the social configuration can be explicitly defined [Bak97]. In order to achieve more effective interactions, these CSCL environments and their resources should be personalized for each participant, depending on its role and group in each activity, and on the objectives that must be reached [Ase08].

Historically, the evolution of CSCL environments (also referred as CSCL systems) started with small isolated software tools or applications like emails, chats, or instant messengers, that were originally designed to facilitate the communication among users [Sta06]. These tools were later arranged into CSCL software environments, which provided several forms of pedagogical scaffolding for collaborative learning, and included additional shared tools like calendars or editors. As an example, C-CHENE [Bak96], whose purpose was to facilitate the learning of concepts in physics through modeling, was one of the first CSCL environments, and included some specific

[3] `http://cat-cscl.eu`. Last visited: June 2012.

collaborative tools like an energy chain editor. However, these first CSCL systems promoted an unstructured collaboration among users, as no tutor could explicitly define the learning objectives, neither a script could be formalized with the sequence of activities that should be realized and the shared objectives that should be achieved in each activity [Dil02a]. The use of scripting to guide learners through collaborative learning situations is a common practice to structure collaboration, and has been demonstrated to increase the effectiveness of interactions and learning among students [Dil02a]. According to authors in [Gom09,Her06a], scripted collaborative learning has a life cycle comprising four phases: the *design* phase, in which educators (or instructional designers) define the computational support, the structure of activities, the learning objectives, the group structure (but not the specific components belonging to each group), and the tasks that tools and resources should support in each activity (e.g. synchronous text editing) [Kop05, Mia05]; the *instantiation* phase, in which educators populate the groups, select the specific tools supporting the activities [IMS03], create the different tool instances [Bot08, Per10] for each group in each activity, and customize the environment according to the needs of each participant; the *enactment* phase, in which students (or learners) carry out the activities defined for the collaborative learning situation, being monitored by educators (or monitors) [Dil07], which can mediate to promote the learning process; and the *evaluation* phase, in which educators (or evaluators) assess the acquired knowledge and skills of students [Dil02a, Vil09].

Next generation CSCL environments allowed the definition of different roles (characterizing, for instance, educators, students and administrators with different permissions in the learning environment), and facilitated the structuring of collaborative learning situations through scripts. Some examples of pioneering CSCL environments that supported the use of roles and scripting are Universanté [Ber01], a specific purpose system aimed at learning about public health problems, and Gridcole [Bot05], a general purpose system that could be tailored to adjust to educators' needs. At that historical moment, the term Virtual Learning Environment (VLE) [App99, Dil00] was coined to define those CSCL systems that, like Universanté or Gridcole, supported a role hierarchy in which the educator was the main actor, and that provided a shared customizable workspace for the realization of structured individual and collaborative activities. Nevertheless, a formal definition for VLE has not been agreed in the community of researchers, and sometimes terms like Learning Management System (LMS), Content Management System (CMS), Learning Content Management System (LCMS), Managed Learning Environment (MLE), or simply, Learning Platform (LP) are used as synonyms of VLE. Interestingly, some papers point out minor differences among these terms. For example, authors in [Dvo11] see VLE and LMS as interchangeably names that refer to software systems designed to support teaching and learning using browsers, and that include tools likes quizzes, wikis, or blogs, while CMS is the term employed to designate centralized data repositories that allow publishing, editing and reading general purpose content.

In this dissertation, a VLE is defined as an educator-centered system that allows the design, instantiation, enactment, and evaluation of collaborative learning situations through a set of synchronous/asynchronous, face-to-face/distance, individual/collaborative learning activities, which are supported by a collection of available tools and resources; this definition is consistent with the most common use of the term VLE in the literature (see for example [Dil02b, Sti07, Wel06, Xu05]). Nowadays, the most commonly used VLE is Moodle[4][5] with more than 66,000 installations in 216 counties as of this writing[6]. Nonetheless, some other outstanding examples of VLEs that educators are largely using worldwide are LAMS[7] (Learning Activity Management System), .LRN[8] (Learn, Research, Network), Sakai[9], Blackboard[10], Claroline[11] or SharePoint LMS[12].

In the last decade, VLEs have quickly become mainstream, especially for distance and blended learning, both in academia [Dun03, Wel06] and industry [Mor03]. However, some practitioners and researchers on the field consider that VLEs are too much focused on meeting the needs of institutions, rather on the own learners' needs [Sev08]. Therefore, a research trend proposing more student-centered software alternatives, which can be grouped under the term Personal Learning Environment (PLE) [Att07, Har06, Wil06], has strongly emerged in the last few years. Despite the term PLE is relatively new, some research works on this topic, like Symba [Bet03], which promoted the personalization of the learning environment by the own students, were published almost a decade ago. On the contrary, some other practitioners and researchers understand that educators should be responsible for providing learners with adequate learning resources and tools, in order to develop intended knowledge and skills systematically [Mue11, Wel07a], as it generally happens in VLEs. Nevertheless, they all agree on the fact that PLEs will not replace VLEs, since both can coexist, or even merge, depending on the learning scenarios and pursued objectives. For example, authors in [Wil06] suggest that PLEs will become dominant on informal learning and competence-based learning, while VLEs will be preferred for formal education. At this point, it is important to note that this dissertation focuses its research scope on VLEs, but the expected contributions that will be defined later in this chapter, could also be useful for PLEs, as well as for other environments that might be employed for collaborative learning, such as wikis [Aug04] or social networking sites [Lip02].

[4] `http://moodle.org`. Last visited: June 2012.
[5] Originally an acronym for "Modular Object-Oriented Dynamic Learning Environment", although this acronym is not used anymore.
[6] `http://moodle.org/sites`. Last visited: June 2012.
[7] `http://lamsinternational.com`. Last visited: June 2012.
[8] `http://openacs.org/projects/dotlrn`. Last visited: June 2012.
[9] `http://sakaiproject.org`. Last visited: June 2012.
[10] `http://blackboard.com`. Last visited: June 2012.
[11] `http://claroline.net`. Last visited: June 2012.
[12] `http://sharepointlms.com`. Last visited: June 2012.

1.1. Research problem of the dissertation

Nowadays, VLEs are one of the most widespread systems for the support of collaborative learning situations [Wel07b]. Most VLEs, such as Moodle, LAMS, Sakai or Blackboard support the features described in [Osu99] to foster learning through collaboration. In this regard, they allow the definition of a social configuration based on roles and groups, the structuring of the situation in activities, courses and/or lessons with predefined learning objectives, and the use of several built-in tools and resources in each activity. Besides, they can also support the four phases defined in the life cycle of scripted collaborative learning [Gom09,Her06a], although only a few of these VLEs (e.g. LAMS) promote the formalization of scripts. Therefore, educators can typically design and instantiate individual and collaborative learning activities within VLEs that students enact afterwards, being their acquired knowledge and skills evaluated by the educators after the completion of the activities. The actual implementation of these four phases (design, instantiation, enactment, and evaluation) depends on the learning strategy and the architecture of the VLE. For instance, Moodle does not explicitly separate design, instantiation and enactment, since it is based on the philosophy of pedagogical bricolage, according to which educators can refine and iterate over the learning design as the activities are being carried out [Ber05]. On the other hand, LAMS explicitly distinguishes between design, instantiation and enactment in three different internal environments (authoring, monitoring and learning environments) aimed at different actors [Dal03]. However, there exists some overlapping in the way these three phases are implemented in LAMS, since the selection of specific tools, which is due at the instantiation phase, is made at design time within the authoring environment.

In any case, those practitioners designing and instantiating collaborative learning situations through VLEs must indicate, at some point, the set of tools they want the learners to employ in order to carry out the activities defined for the situation. VLEs typically include a limited set of ten to twenty built-in tools, both for individual or collaborative purposes. These tools can be added to the learning activities, in order to facilitate students the completion of the intended tasks. Some examples of tools that appear in the distribution of the main VLEs are chats, forums, notice boards, questionnaires and polls [Col07,Dal03,Uzu06]. However, the actual implementation and functionality offered by each of these tools also depends on the specific VLE; for instance, the implementation of the Moodle chat differs from that in the LAMS chat. Built-in tools are usually designed for general purpose tasks and so, they can support common learning activities in different learning contexts. Nevertheless, the reduced set of VLE built-in tools is frequently criticized by educational practitioners, who consider it an important limitation to support a wide range of learning activities [Bow11,Dag07,Fie07,Liv08].

Apart from the evidences of this problem found in the literature, a set of interviews performed as a preliminary work for this dissertation helped to gain insight into the reasons educators

have to be unsatisfied with VLE built-in tools[13]. One of the reasons educators argue is that they expect to find in VLEs the same tools they usually employ in their classes, such as generic presentation tools, or specific simulators and processors (e.g. "*I would like to have a shared whiteboard and a network simulator in my VLE for my Advanced Networking course*"). The motivation for this assertion comes from the reduced number of available tools in VLEs, which is even more reduced when specific purpose tools are required. In some other cases, the functionality of the VLE built-in tools does not come up to the expectations of educators (e.g. "*Moodle allows me to link a web content as a resource, but I cannot configure this resource in a different way for each of the groups I need to define in my course*"). In addition, some practitioners are used to certain concrete tools, favoring them ahead of VLE built-in tools (e.g. "*I frequently employ Google Spreadsheet*[14] *to create my spreadsheets, and so, I prefer it to the equivalent spreadsheet tool in LAMS*"). The educator that made this comment also stated that he is sometimes unwilling to try new tools because of the extra time and effort that may take. Finally, educators can be managing information and contents in other applications or services, thus requiring an easy way to include them in their regular VLE (e.g. "*I manage a MediaWiki*[15] *distribution in which I prepare my master courses, and I would like to link its content to our institutional VLE, Moodle, instead of using the default Moodle wiki*"). This educator also mentioned that a single centralized platform for the enactment of his lessons and the supplying of extra material is essential to deliver his courses. The conclusion of these interviews together with the cited references in the literature is that the restricted set of available built-in tools in existing VLEs hinders and even precludes from designing, instantiating and enacting many individual and collaborative learning activities.

The problem with the restricted set of built-in tools in learning platforms might seem recent, but it is not new at all. In fact, three different research lines have been historically proposed to overcome this problem. Pioneering works aimed at developing new flexible and tailorable VLEs, which were specifically designed to facilitate the addition of external tools. This is the case, for instance, of DARE [Bou01], Symba [Bet03] or Gridcole [Bot05, Bot08], and more recently, Pelican [Vel09]. The main limitation of these new VLEs was that they were conceived to replace other existing and popular VLEs. Unfortunately, educators and students that are used to a given VLE in their classes are typically reluctant to embrace a new one, sometimes because of the learning effort and the adaptation period required, and sometimes because their institutions force them to use a particular, institutional VLE [Ala10a]. In addition, these flexible and tailorable VLEs were the outcomes of research projects that never become stable products, thus hindering their adoption by those potentially interested. Some other authors decided to

[13] The comments quoted in this paragraph were collected from semi-structured interviews with six educators with extensive experience in learning design and collaborative learning, working in the School of Telecommunication Engineering at the University of Valladolid (Spain). This comments are employed as additional evidences to support the general discontent of practitioners with VLE built-in tools

[14] `http://docs.google.com`. Last visited: June 2012.

[15] `http://mediawiki.org`. Last visited: June 2012.

develop tools from scratch for certain VLEs. This is the case of many Moodle modules and plugins[16] [Gut09] or Blackboard Building Blocks [Pit03] that extend the set of available tools in Moodle and Blackboard, respectively. These tools may provide the functionality expected by some educators combined with the native VLE features, and occasionally, they have been adopted by the own VLE providers, and even packaged in subsequent official releases of these VLEs. Nevertheless, those educators that normally employ other VLEs, such as LAMS or Sakai, cannot add Moodle plugins or Blackboard Building Blocks for the support of their learning activities. Besides, these tools might also be replacing existing tools with a similar functionality, and so, once again, educators and students should assume an additional learning effort and adaptation period to master these newly developed tools. Both alternatives resulted in a lack of widespread adoption of related works, causing the quick rising of a third alternative: **the integration of existing external tools in existing VLEs** [Fon09, Fue11, Sev08].

The integration of existing external tools in existing VLEs aims at offering practitioners a larger set of available tools in their commonly used VLEs for the support of their learning activities [Fue11]. Researchers and developers working on this line are especially encouraged by the recent spread of web technologies [Pau08] and the growth of Web 2.0 [ORe07], which brought an explosion of third-party software tools used more and more by practitioners, in principle, outside of VLEs [Wel07b]. Furthermore, many of these tools are freely available for education (and in some cases for any other use), which makes them very interesting for schools, colleges or universities that cannot afford multiple software licences for commercial applications. Examples of freely available tools are Google Apps[17], Twitter[18], Wordpress[19], Flickr[20], or Doodle[21]. A clear sign of the success of software tools in education is the publication of lists with the most useful tools that educators are employing in their classrooms. In particular, the Centre for Learning & Performance Technologies (C4LPT) updates every year its well-known Top 100 Tools for Learning site[22] with the most outstanding educators' preferences regarding applications, web sites, learning platforms and hardware devices. Another example is the Cool Tools for Schools wiki[23], which indexes a lot of Web 2.0 tools, arranged according to their main educational task (e.g. drawing tools, mapping tools, audio tools, etc.). The huge number of software tools available for education, the popularity of VLEs, and their limitations regarding built-in tools, motivate the significant number of research works that are tackling the integration of existing external tools in existing VLEs [Bol07, Boo09, Bla09, Dod08, Fon09, Fue11, IMS06c, Sev08].

[16] `http://moodle.org/plugins`. Last visited: June 2012.
[17] `http://google.com/apps/intl/en/edu`. Last visited: June 2012.
[18] `http://twitter.com`. Last visited: June 2012.
[19] `http://wordpress.org`. Last visited: June 2012.
[20] `http://flickr.com`. Last visited: June 2012.
[21] `http://doodle.com`. Last visited: June 2012.
[22] `http://c4lpt.co.uk/recommended/2011.html`. Last visited: June 2012.
[23] `http://cooltoolsforschools.wikispaces.com`. Last visited: June 2012.

Nevertheless, the integration of a tool in a VLE is not an easy task mainly due to two reasons. First, each VLE and each tool typically imposes different heterogeneous requirements to enable their functional extension and technological interoperability with other systems. These requirements are included in the *integration contracts* [Ghi06, Lar02], being these contracts explicit or not. An integration contract determines, at least, the technologies, the interfaces, and the data models that must be employed to enable the communication between the system that imposes such contract and other applications [Ala12a]. In general terms, the more requirements defined in the integration contracts, the tighter the integration, and the richer the communication that can be established between VLEs and tools; as opposed to contracts with few requirements that promote loose integrations, but normally at the expense of a poorer communication between VLEs and the tools [Pau09]. Second, a *developer* must program the code needed to enable the communication between a VLE and a tool contract. The role of developer can be played by anyone interested in that integration; this may include the VLE provider, the tool provider, or a third-party [Ala10a]. Those programming this code will usually expect a benefit in return. This benefit could be recognition, reputation, economic compensations, or the satisfaction to use (or let others use) the integrated tools within VLEs. Research works tackling the integration of external tools in VLEs should thus consider in their proposals the expected developers; otherwise, it might happen that nobody develops the aforementioned code.

The development of the code that enables the interoperability between a VLE contract and a tool contract thus demands a certain **development effort** [Alb83]. Nevertheless, this development effort is significantly high in most integration approaches, mainly due to two factors. First, many of these approaches foster a *one-to-one* integration between VLEs and tools. This implies that new code must always be developed for each new VLE-tool integration, as it happens with most of the Moodle plugins aimed at integrating existing external tools in this VLE. For example, if a developer assumes the effort to integrate Flickr in Moodle using Moodle's own extension mechanism (i.e. the Moodle integration contract), he could barely reuse either the generated code or the acquired knowledge, when integrating Flickr in LAMS, Sakai or Blackboard. Second, many approaches promote a *tight integration* [Ort90] between VLEs and tools. This requires generating an important amount of extra code aimed at enabling richer interactions among them, even if these interactions are not necessary for the support of most learning situations. The IMS Learning Tools Interoperability (IMS LTI) specification [IMS06c] (popularly referred as to Full LTI) and the CopperCore Service Integration (CCSI) framework [Vog06] are two examples of integration proposals that entail a high development effort because of this second factor. The lesson that can be learned from these and other similar works is that a high development effort limits the adoption of integration approaches, since it may discourage developers to contribute to the integration of new tools and VLEs, also reducing the interest of practitioners and institutions on these approaches.

Trying to address these issues, two works were recently proposed with the aim of reducing the development effort, by fostering a *many-to-many integration* between VLEs and tools, and following *loosely-coupled* approaches. This is the case of Apache Wookie (Incubating)[24], a reference implementation of the software architecture proposed in [Wil08]. Apache Wookie enables the integration of different software applications, provided that they are developed following the W3C (World Wide Web Consortium) Widgets specification [W3C11]. Nevertheless, this can be considered a very **strict technological restriction** that hinders the possibility of integrating many other existing tools that do not fulfill this restriction (e.g. Google Apps, Wordpress, etc.), although these tools could be of great interest to support many learning activities. Therefore, this restriction reduces the set of external tools integrated through Apache Wookie that educational practitioners might use in their learning situations. Besides, the W3C Widgets specification was defined to standardize simple tools or mash-ups, thus also limiting the functionality of the external tools that can be integrated. The conclusion is that the imposition of strict technological restrictions, like those in Apache Wookie, may discourage institutions and practitioners from adopting this and other approaches that also present this limitation.

Another work that has recently been proposed with the aim of reducing the development effort, by fostering a *many-to-many integration*, and following a *loosely-coupled* approach, is IMS Basic LTI (Basic Learning Tools Interoperability) [IMS10b]. Basic LTI is a specification defined by the IMS GLC (IMS Global Learning Consortium) as a subset of the aforementioned Full LTI [IMS06c], although they both were merged this year in one single specification called simply LTI [IMS12]. Basic LTI enables an easier integration of a wide range of existing external tools in existing VLEs. However, Basic LTI also presents an important limitation: it just enables the retrieval of a generic instance for each external tool. As a consequence, Basic LTI does not allow educators to request a separate creation and particularization of tool instances for each group defined in each learning activity, not being responsible for the management of the user and group access to the functionality and content of external tool instances. Therefore, Basic LTI cannot take advantage of the collaborative features provided by VLEs, such as the management of groups to set the social configuration during the design and instantiation of collaborative learning situations [Mar04, Osu99]; nor the personalization of the integrated tools, in order to achieve more effective interactions among students during the enactment of these situations [Ase08]. In other words, Basic LTI is a very limited integration approach **to support the instantiation and enactment of collaborative learning situations**. This limitation can motivate that many educational practitioners and institutions that promote the collaboration among learners in their practices may discard Basic LTI or other similar integration approaches that also present this limitation.

[24] `http://incubator.apache.org/wookie`. Last visited: June 2012.

Considering all these issues, current integration works present three important limitations that hinder their widespread adoption for the support of collaborative learning situations. A solution that could overcome these limitations would be the definition and implementation of a middleware integration architecture [Bri04] that: fosters a *many-to-many integration* and follows a *loosely-coupled* approach in order to reduce the development effort; imposes only *restrictions that most VLE and tool providers currently meet*; and *facilitates the instantiation and enactment of both individual and collaborative activities* that require the integration of external tools, by enabling the creation and configuration of external tool instances according to the social configuration defined in VLEs. Besides, this architecture should be compatible with other existing loosely-coupled integration approaches, so that the adoption of one of them does not preclude from using the others. Finally, and as an historical note, it is noteworthy that some of the main ideas regarding tool integration through loosely-coupled approaches emerged at the heart of the GSIC-EMIC[25] research group [Ase08, Bot08] in which this dissertation is being developed, and have been confirmed by some of the aforementioned recent integration works like Apache Wookie and Basic LTI.

1.2. Objectives and contributions

The previous section described the three main limitations of current research works addressing the integration of external tools in Virtual Learning Environments. These limitations are mainly responsible for the lack of widespread adoption of existing integration works in this context. In order to overcome these limitations, the global objective of this dissertation is:

To design, develop and evaluate a middleware architecture that enables the integration of multiple existing external tools in multiple existing VLEs, demanding an attainable development effort to integrate new VLEs and tools, imposing only basic restrictions that most VLE and tool providers already meet, and supporting enough functionality to facilitate the instantiation and enactment of collaborative learning situations.

In order to achieve this global objective, several partial objectives are set. These partial objectives, as well as the specific contributions expected for each of them, are depicted in Figure 1.1 in the general scheme of this dissertation, and are described next:

- *To analyze the problem of integrating existing external tools in existing VLEs.*

  This partial objective is intended to identify and analyze the main requirements of the stakeholders involved in the integration problem, as well as the main technological and

[25] `http://gsic.uva.es`. Last visited: June 2012.

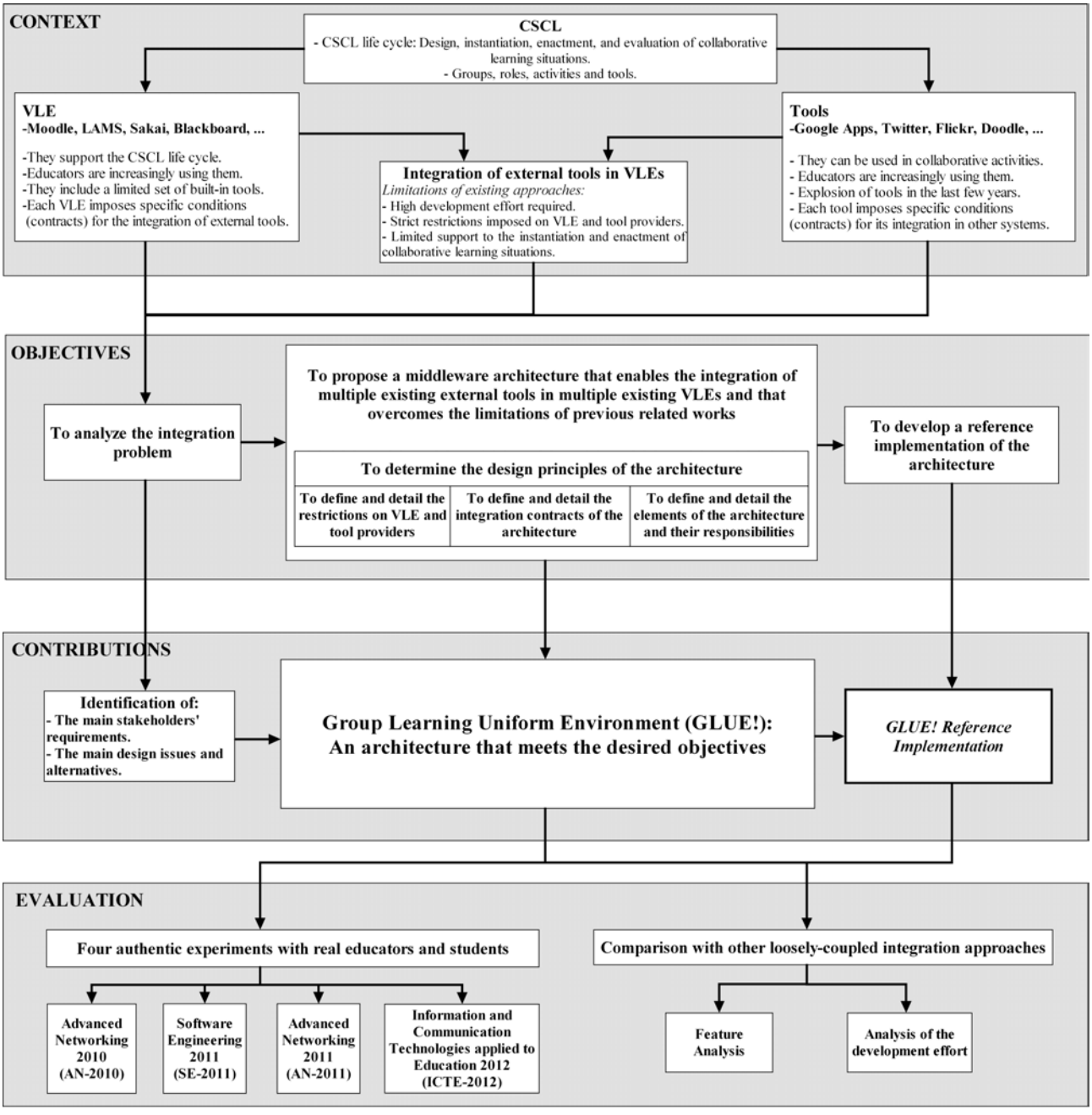

Figure 1.1: General scheme of the dissertation including its context, the aimed objectives, the expected original contributions, and the evaluation planned to assess them.

functional design issues that should be taken into account before proposing new integration approaches. This analysis is complemented by a deep study of the integration works in the literature, which include some middleware architectures, and also some standards defined by international organisms aimed at establishing new generic integration contracts.

On this objective, the contributions of this dissertation are: the **identification of the main stakeholders' requirements**, and the **identification of the main design issues and alternatives to be considered when proposing new integration approaches**. Both the stakeholders' requirements and the alternatives to the design issues are taken into account when addressing the global objective of this dissertation. Actually, these requirements and issues are the guidelines to design and develop the architecture proposed in this dissertation. The conclusions about this partial objective and the contributions have been published in [Ala10a], including the definition of the main issues and the integration proposals up to 2009, being revised and updated in [Ala10c] to include later integration works. Besides, the stakeholders' requirements have been formally embodied in [Ala12a].

- *To propose a middleware architecture that enables the integration of multiple existing external tools in multiple existing VLEs, and that overcomes the limitations of previous related works.*

  This partial objective consists on the proposal of an architecture that connects VLEs and tools, whose design principles follow the guidelines distilled in the previous partial objective, and that overcomes the three main limitations of previous related works: the high development effort, the strict restrictions on VLE and tool providers, and the limited support to the instantiation and enactment of collaborative activities. According to these limitations, three parts must be clearly distinguished in the proposal of this architecture:

  - To define and detail the *restrictions on VLEs and tools*, provided that these restrictions must be met by most existing VLE and tool providers.
  - To define and detail the *integration contracts of the architecture*, provided that these contracts must reduce the development effort required to integrate existing external tools in existing VLEs.
  - To define and detail the *elements of the architecture and their responsibilities*, provided that these elements must facilitate practitioners the instantiation and enactment of collaborative learning situations that requiere the integration of external tools within VLEs.

  The contribution of this dissertation to this objective is **GLUE! (Group Learning Uniform Environment), a middleware integration architecture that meets the desired objective**, thus overcoming the limitations of previous related works. The proposal of this architecture is detailed in [Ala12a].

- *To develop a reference implementation of the proposed architecture.*

  This partial objective aims at developing a reference implementation that includes the structure and the main functionality of the proposed architecture. Also, as part of this objective, a start-up collection of VLEs and tools is generated. This reference implementation should be a model to facilitate external contributors and third-party developers the integration of new VLEs and tools. Nevertheless, it is important to remark that the main ideas underlying the proposal of the GLUE! architecture could be applied to other implementations, different from this reference implementation.

  The contribution of this dissertation to this partial objective is **GLUE!-RI (GLUE! Reference Implementation**), the reference implementation of the GLUE! architecture. This implementation and its prototype have been published in [Ala12a], with further details for the particularities of the VLE LAMS in [Ala11b]. Besides, [Ala12c] is a demonstration paper showing the usage of this reference implementation in Moodle and LAMS. The code of GLUE!-RI is available at `http://gsic.uva.es/glue` to be downloaded and installed by anyone interested.

- *To evaluate the proposed architecture and its reference implementation.* Once the proposal has been defined and developed, it must be evaluated. This evaluation must assess whether the architecture meets the stakeholders' requirements, and overcomes the limitations of previous related works. Besides, since the objectives and contributions belong to both the CSCL and software integration fields, then both the educational and technological dimensions should be considered in this evaluation [Zel02]. Taking these considerations, the evaluation of this proposal comprises:

  - *Four authentic experiments with real educators and students* [Dew01] at university level. These experiments serve to study the compliance of the architecture to the stakeholders' requirements. Besides, they respond to educators' needs in different learning settings, and cover several knowledge domains, collaborative strategies, VLEs and external tools. Different versions of the GLUE! reference implementation, which is incrementally and iteratively developed, are employed by practitioners in these four experiments. The CSCL-EREM (Computer Supported Collaborative Learning Evaluand-oriented Responsive Evaluation Model) [Jor09], which is a framework that facilitates the formal evaluation of software systems in CSCL settings supports the evaluation of GLUE! in these experiments. Data from the usage of the architecture are collected combining qualitative and quantitative techniques [Kit96a], and analyzed using the mixed method proposed in [Mar03].
  - *A comparison of GLUE! with other loosely-coupled integration approaches.* This comparison includes a *feature analysis* [Kit96a], and an *analysis of the development effort*

required for the integration of new tools and VLEs. The latter is based on an empirical measure, called the source lines of code (SLOC), which is frequently used as an indicator of software complexity [Alb83].

This evaluation has partially been published in [Ala12a], showing the results and conclusions of the four authentic experiments, as well as the comparison of the development effort in loosely-coupled integration approaches. Besides, an extended evaluation focused on two of the experiments can be found in [Ala12b].

1.3. Research methodology

This dissertation is framed into a multidisciplinary field, since its objectives and contributions are oriented to the design and development of a software system, which is expected to have a certain impact on the educational domain. This is the main reason that motivates an evaluation that combines both the educational and technological dimensions [Zel02], as already anticipated in the previous section. Similarly, the research methodology should also combine different methods to better take into account the multidisciplinary nature of the contributions. In this regard, the engineering method proposed by Adrion [Adr93] could cover most of the methodology needed for this research. This method iterates on the following four steps: observe existing solutions, propose a better solution, build or develop, measure and analyze. Nevertheless, this method could be merged, especially in the evaluation stage, with the empirical method [Big82], due to the employment of experiments [Dew01] as part of the educational evaluation. Glass [Gla95] formalized the combination of these two methods, taking Adrion's work as a basis, in four iterative phases that have already been applied to similar recent dissertations with technological contributions in the CSCL field [Her07b, Per11, Vil10]:

- **Informational phase**. The first phase aims at collecting information about the domain in which the research is carried out. The first task is the review and analysis of existing literature in order to detect the affordances that may become research questions, and the related works that previously explored such affordances. Next, it is convenient to participate in scientific events and research projects to assess the potential interest of these affordances, and also to get feedback from the community of people working in the field. In this dissertation, this phase addresses the review of multiple works on TEL, CSCL, learning platforms, use of tools for educational purposes, technological trends, software engineering, software architectures, and software integration in multiple contexts. During this process, the integration of external tools in VLEs to facilitate the realization of a wider range of collaborative learning situations emerges as the main research question for this dissertation. Besides, this phase also includes the participation on European, national

and regional projects related to the domain, the attendance to several conferences, and the realization of short research stays with experts on software integration and learning platforms.

- **Propositional phase**. The second phase aims at proposing or formulating hypothesis or solutions to the research questions identified in the informational phase. In this regard, the literature review allows to identify the limitations of previous related works on this topic and raises new original alternatives to tackle these limitations. In this dissertation, this phase comprises the identification of the main stakeholders' requirements, the analysis of the main design issues that should be considered when tackling the integration problem, and the proposal of a middleware architecture that takes into account these requirements and issues in order to overcome the limitations of previous approaches.

- **Analytical phase**. The third phase aims at analyzing and exploring the propositions made in the previous phase, to achieve a demonstration or formulation of principles. This may include the development of the necessary systems or applications that facilitate this demonstration. In the context of this dissertation, a reference implementation of the architecture proposed in the second phase is developed, providing a prototype that can be tested and used by educational practitioners. This reference implementation is employed for the analysis and evaluation of the contributions of this research, and can be enhanced as the iterations along the phases of this methodology happen.

- **Evaluative phase**. The last phase aims at evaluating the propositions by means of experimentation or observation, helped by the systems or applications developed in the analytical phase (if applicable). In this dissertation the architecture is evaluated by means of four authentic experiments intended to demonstrate that it meets the stakeholders' requirements and overcomes the limitations of previous related works. A supporting comparison with other loosely-coupled integration works is also carried out as part of the evaluation phase.

1.4. Structure of the document

The rest of the document is structured according to the partial objectives defined in section 1.2. Therefore, after this introduction, chapter 2 delves into the theoretical background of this research. That includes the main ideas on which the CSCL research field is based, emphasizing the benefits of using learning platforms and software tools for the support of interactions among learners. Besides, this chapter reviews and compares the main Virtual Learning Environments and other outstanding platforms, as well as a set of representative tools employed for educational purposes. Then, a deep analysis of the integration problem in this context is carried out, with the aim of identifying the main stakeholders' requirements and the main issues that should be

taken into account when designing new integration approaches. Finally, the literature is critically reviewed, indicating which of the detected issues contributed to the lack of success in terms of adoption of current integration works.

Chapter 3 introduces GLUE!, the middleware architecture that enables the integration of multiple existing external tools in multiple existing VLEs. This chapter begins by highlighting the main design decisions of GLUE!, which are the result of taking into account the lessons learned from the previous analysis of the integration problem. Then, a global overview of the architecture is presented, detailing the integration contracts and the elements involved, as well as their responsibilities in order to provide the functionality needed to facilitate the instantiation and enactment of collaborative learning situations that require the integration of external tools within VLEs. This chapter also deals with the security issues that appear in the interactions throughout the architecture. Besides, the opportunities for connecting GLUE! and other existing loosely-coupled integration works are discussed before the conclusions.

Chapter 4 presents GLUE!-RI, the reference implementation of the GLUE! architecture. This chapter delves into low level details of each of the GLUE! elements, discussing technical alternatives for their implementation, and pointing out the differences that appear when integrating some outstanding VLEs and tools. Besides, it is in this chapter where different processes related to the architecture are described: from the development of code that an interested contributor should be aware of, through the installation and configuration by a GLUE! administrator, to the usage of GLUE! by educators and learners

Chapter 5 explains and discusses the evaluation of the GLUE! architecture, supported by its reference implementation. This chapter starts by introducing the overall evaluation methodology, including the data gathering techniques, and the data analysis method employed. Then, the four authentic experiments supporting the GLUE! evaluation are detailed. After that, results obtained from these experiments are used to support the conclusions regarding the compliance of the GLUE! architecture to the stakeholders' requirements. Finally, the comparative evaluation with other loosely-coupled integration approaches, focused on the functionality offered and in the development effort required, is presented.

Chapter 6 draws together the conclusions and the future work of this dissertation summarizing the main original contributions and the objectives achieved. In addition, this chapter also links these contributions with a series of active ongoing research works at the heart of the GSIC-EMIC research group. The global objective of all these works is to design and develop an extensible architecture for the support of the main actors involved in the design, instantiation, enactment and evaluation of collaborative learning situations, using GLUE! as the core for the integration of tools in learning platforms, and following many of the design principles that emerged as a consequence of the realization of this dissertation.

This document also includes five appendices that complement the discussion of the dissertation. Appendix A contains more details about the study of the development effort. Appendix B details the data format used for the communication between the elements of the GLUE! architecture. Appendix C collects the documentation of GLUE! for developers interested in contributing to the integration of new external tools. Appendix D gathers further information for GLUE! administrators, including the installation and configuration manuals for the main components of the reference implementation. Appendix E includes user manuals for educators and students using GLUE! within three representative VLEs and learning platforms: Moodle, LAMS and MediaWiki.

Chapter 2

Integration of external tools in VLEs

The aim of this chapter is to present the theoretical background on which the research work of this dissertation is based, emphasizing the specific challenges it undertakes. The chapter starts by introducing in section 2.2 the CSCL as a multidisciplinary research field, in which technology is employed to foster the interactions and collaboration among learners. Then, VLEs are studied in section 2.3 as significant examples of successfully adopted systems that facilitate the design, instantiation, enactment and evaluation of collaborative learning situations. The use of software tools alongside the learning process is introduced in section 2.4, discussing the tool life cycle when supporting collaborative activities. In this point, special emphasis is given to the limitation regarding the restricted set of VLE built-in tools, and how this limitation, together with the massive use of third-party external tools, has caused a great research interest in the learning community, resulting in many works proposing approaches that address the integration of external tools in VLEs. Section 2.5 discusses the integration problem in this context, identifying the requirements of the main stakeholders (practitioners, developers, and providers), as well as the main design issues and alternatives that should be considered by those tackling this problem in order to increase their chances of adoption. These design issues are: the number and kinds of restrictions imposed on VLEs and tools, the coupling and multiplicity promoted by the integration, and the degree of functionality offered. Next, Section 2.6 presents the most representative integration works, relating them with these design issues, and discussing the reasons for the lack of success in adoption of these works. Finally, section 2.7 distills some lessons learned from the earlier discussion of the integration problem and the existing works. These lessons should be of interest for those proposing new integration approaches, including the one presented later in this dissertation.

The identification of the main issues that should be taken into account when tackling the problem of integrating existing external tools in existing VLEs is a contribution of this dissertation, and has been published in [Ala10a], including the definition of these issues and an analysis of existing proposals up to 2009. These issues were revised and updated in [Ala10c, Ala12a] to include later integration works.

2.1. Introduction

TEL is a field of study that aims at facilitating and improving teaching and learning through technology [Joh04]. Research on TEL covers all educational levels in both formal and informal learning. It is important to note that the term TEL is sometimes misused as a synonym of distance learning or e-learning. On the contrary, TEL also includes face-to-face, and blended learning scenarios in which technology is employed [Osg03]. With the introduction of technology into learning designs based on traditional pedagogies new challenges arise, requiring a redesign of educational systems and practices [Kop04].

CSCL is the multidisciplinary field within TEL that studies how technology facilitates the collaboration in groups, in order to increase the effectiveness of learning [Kos96]. In collaborative learning situations, the acquisition of knowledge and skills is carried out through interactions with peers [Dil99]. These interactions have been demonstrated to promote a more effective and deeper learning, compared to other processes, such as individual or competitive learning [Vig78], especially regarding high-level reasoning and knowledge transfer. CSCL environments are those systems that include a set of features, resources and software tools intended to support the realization of collaborative learning situations which, remarkably, may include both individual and collaborative activities [Osu99].

VLEs, such as Moodle, LAMS, Sakai or Blackboard, are successful examples of centralized software systems used worldwide in TEL, especially for distance and blended learning. Besides, VLEs are recurrently used by practitioners to support collaborative learning processes, since they enable the definition of social configurations of users, structures of activities, and the use of some built-in tools that promote collaboration and groupwork (e.g. chats, forums and wikis) [Wel07b]. All these features make VLEs suitable for the design, instantiation, enactment and evaluation of collaborative activities in learning contexts [Jon05].

A large number of studies report the benefits of educators and students using VLEs for learning contexts [Kat10]. For instance, Koeber [Koe05], after carrying out a comparative study between two groups of almost one hundred of sociology students each, found that average grades of the group using a VLE during the course were considerable better that those of the other group, which did not use any technological support. Koeber justified the results obtained asserting that the use of a VLE had a positive effect in the motivation of the students enrolled in that course. Another study by Patzold [Pat05] confirms this assertion by showing an increasing engagement on students' behavior when realizing VLE built-in tool-mediated activities, including tests and discussions on forums, which also had a positive effect on students' final grades. Regarding educators, Seaman [Sea09] found that the majority of faculty members thought that learning outcomes using VLEs were "*as good as or better that through face-to-face instruction*". Finally, O'Leary [OLe02] points out two additional advantages of using VLEs. First, educators can

provide a more flexible support to students. This is due to the fact that educators do not need to be in a fixed time or place to interact with students. Besides, students become more active and independent, making use of online access to learning materials and collaborative features, on and off their institution. Both advantages are essential to support the new teaching models at all levels and contexts.

The advantages of including VLEs as the technological support in education in order to promote a more engaged and structured collaborative learning seem clear now. Nevertheless, other studies pointed out some important limitations that should also be considered. For instance, Bower [Bow11] reported educators' concerns regarding potential technical problems or crashes that may happen when their courses are entirely supported by VLEs. Nevertheless, this problem has not stopped thousands of institution from adopting VLEs, and here it is noteworthy that any technological support always implies some risk of failures at first, but after improvements and tests, the chances of occurrence of failure are minimized. The same study also remarked that setting learning situations and courses within VLEs is a time-consuming and burdensome task for educators (compared to traditional face-to-face courses). Nevertheless, the introduction of new software systems normally entails a certain initial learning effort (i.e. learning curve), but in the case of VLEs, once this effort is undertaken, educators can take advantage of VLEs functionality in order to reuse educational content and automatize the management of courses and lessons [Dal03]. Other authors [Att07, Har06, Sev08] go beyond, criticizing the institutional and educator-centered basis of VLEs, targeting their research towards other platforms, generally called PLEs [Har06], in which learning is controlled by the own learners. Nonetheless, these authors agree that both educator-centered and learner-centered models can coexist, being VLEs dominant platforms in formal education [Wil06].

The small and reduced set of available built-in tools is also agreed as another important limitation in VLEs [Con10, Dag07, Liv08]. For instance, practitioners on Bower's study [Bow11] stress the "*smaller range of tools*" available in Moodle, as well as the low flexibility regarding LAMS tools. Practitioners' perception on this study may be motivated by their expectations of finding in VLEs the tools they usually employ in their classes. The lack of a larger and proper set of built-in tools thus limits the kind of learning situations that educators may design and enact within VLEs. Furthermore, this limitation might be partially responsible for many educators using VLEs as simple document repositories, in which students mostly perform administrative tasks, such as reading announcements, or submitting coursework [Kat10], rather than carrying out more complex and structured learning activities using the required software tools.

Meanwhile, practitioners are using more and more software tools outside of VLEs, not only in their classes, but also in their daily life. Some reasons supporting this massive tool adoption are: the high number of hardware devices in the market, from laptops to mobiles or tablets; the possibility of an online connection, everytime, everywhere; and the existence of

many software tools offered as Software as a Service (SaaS) applications [Pap03], and designed under Web 2.0 principles [ORe07]. These principles are typically associated to interoperable web-based systems, provided by different vendors, and designed to promote the collaboration and the sharing of information and contents [ORe07]. Such a large number of available distributed tools might overwhelm inexperienced (and even some expert) practitioners, which could benefit from the centralization of these SaaS, Web 2.0 and many other tools in a single environment that already provides educational and collaborative features, like the widespread VLEs.

Summarizing what this introduction has anticipated, the remaining of this chapter focuses on the integration of existing external tools in existing VLEs, in order to enrich the collaborative learning situations that can be carried out within these VLEs. Therefore, classical challenges in software interoperability are considered from the perspective of VLEs and software tools, in the current more and more web-like technological world, in which human collaboration to achieve a common goal (being it learning in this context) is a recurrent process that normally brings better outcomes. An scheme relating the concepts introduced in this section and further explained along this chapter is presented in Figure 2.1.

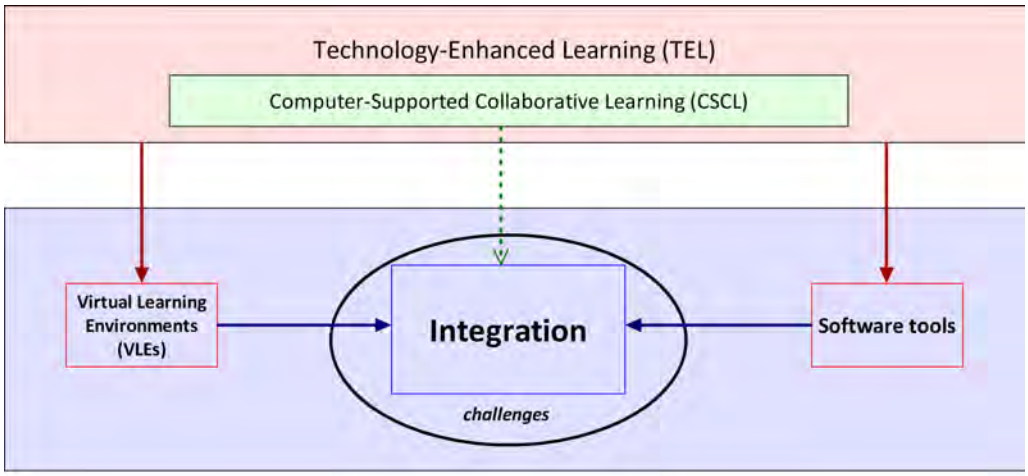

Figure 2.1: Scheme of the main concepts belonging to the research context.

2.2. Computer Supported Collaborative Learning

CSCL is a recent emerging research field within TEL [Kos96], characterized by its multi-disciplinarity [Sta10], involving practitioners from pedagogy, education, computer science, and social science. CSCL origins date back to a NATO-sponsored workshop that was held in 1989 in Maratea (Italy), and which, for the first time, included the term *Computer-Supported Collaborative Learning* in its title [Sta06]. From then on, an increasing global community of researchers

have been working on this field, meeting every two years since 1995, in the biennial CSCL conference, which had its 9th occurrence in 2011 in Hong Kong (China). Besides, this community of researchers have been publishing their novelties and contributions in a specific journal named the International Journal of CSCL (ijCSCL). The evolution of CSCL has been reviewed several times by important names, and can be consulted in the works of Dillenbourg [Dil09] and Stahl [Sta06].

CSCL can be contrasted with other three earlier TEL fields identified by Koschmann in his highly cited book [Kos96]. These fields are sequenced as follows: CAI (Computer-Assisted Instruction), ITSs (Intelligent Tutoring Systems), and Logo-as-Latin. CAI began in the 1960s and conceived learning as a completely passive process where information had to be memorized, and where the role of educators was just acquiring a prior knowledge that was later passed on and shared with the students. ITSs started in the 1970s and relied on the personalized tutoring, creating artificial intelligence applications that represented student mental models, and that responded with appropriate feedback to students' interactions. Logo-as-Latin gained momentum in the 1980s under a constructivist approach, and encouraged students to explore, develop, and run their own applications, thus building their knowledge by themselves. Finally, CSCL began in the 1990s motivated by social constructivism [Jon99, Win93], and promotes social interactions among students and learning together opportunities using technology, being educators only mediators that facilitate (if needed) these interactions.

The support of collaboration processes through computers and technology is not restricted to the CSCL field, since it is also the grounding of CSCW (Computer-Supported Cooperative Work) [Ell91,Gru92]. Nevertheless, despite sharing some important features, such as the promotion of social interactions and groupwork, CSCL and CSCW target completely different goals. While, CSCL pursues more effective learning outcomes through the collaboration in groups, CSCW pursues the increase of productivity in industrial and corporate environments through the cooperation in groups. Apart from the differences in the acronyms regarding "learning" and "work", which suggests different goals and approaches, the differences between the terms "collaborative" and "cooperative" can be further discussed. Actually, even though most authors consider these two terms rather synonyms, others point out that collaboration requires realizing all the tasks together (the focus is on the process), while cooperation may entail a division of work into independent sub-tasks (the focus is on the product) [Dil99]. Nonetheless, the terms collaboration and collaborative learning are preferred along this dissertation, and they are used indistinctively when embracing cooperation.

All in all, CSCL is a multidisciplinary research field that devotes important efforts to the development of applications and systems for the support of collaborative learning situations. Some of these systems are explored along this chapter, as well as their technical, functional and pedagogical challenges, in order to improve their support to the life cycle of collaborative learning situations.

2.2.1. Life cycle of collaborative learning situations

The life cycle of collaborative learning situations [Gom09, Her06a], being they supported by technology or not, involves the whole process starting from the first fuzzy ideas in educators' minds regarding a potential structure of individual and collaborative activities (and their settings), to the obtention and subsequent evaluation of students' outcomes after the realization of these activities. This life cycle is sometimes made explicit through the use of scripting [Her07a, Vil09], which aims at "facilitating social and cognitive processes of collaborative learning by shaping the way learners interact with each other" [Har07b]. Numerous studies have shown the benefits of scripting; among them, the greater effectiveness of collaboration and learning [Dil02a, Her07a]. Nevertheless, in some other cases, the life cycle of collaborative learning situations is not formalized through scripting, thus promoting more open and flexible designs motivated by practitioners' improvisation [Pri11]. This life cycle (explicit or not) can be found in most formal learning situations, as a juxtaposition of non-curriculum-based, unstructured, unsequenced, and non-assessed informal situations [Wel91]. Nonetheless, cautions must be taken, since the formal/informal border is sometimes blurred [Dil09]. Thus, this life cycle could be found in other situations different from those belonging to formal learning.

The life cycle of collaborative learning situations comprises four phases: **design**, **instantiation**, **enactment** and **evaluation** [Gom09,Her06a], as shown in Figure 2.2. In the first phase (design), skilled instructional designers [Vil09], or simply any educator, define the structure or sequence of individual and collaborative activities, with the tasks that participants have to accomplish, as well as the learning objectives and the generic social configuration. Significantly, this first stage does not consider specific participants, nor specific tools, thus promoting the reuse of learning designs in other scenarios and the creation of design templates [Bou06] that can be shared among educators. In the second phase (instantiation), those educators supervising the collaborative learning situation select the tools and populate the groups according to the list of students participating in the situation, particularizing this way an abstract generic design into a specific context. Next, in the third phase (enactment), students carry out the learning activities, being monitored by educators. Finally, in the last phase (evaluation), educators evaluate students' work as well as their acquired knowledge and skills [Dil02a]. It is noteworthy that the results of the evaluation phase give feedback to educators, who can improve and adapt their learning designs to achieve better outcomes, as it is represented in Figure 2.2.

Other works in the literature define life cycles for collaborative learning situations with small variations. This is the case of Vignollet [Vig08], who suggests a linear life cycle composed of three phases: modeling, operationalization and execution. This life cycle emerges from applying the three levels defined in the IMS LD specification [IMS03] for the definition of collaborative scripts. Nevertheless, the three phases defined by Vignollet can be easily mapped to those of design, instantiation and enactment (in this order) according to the life cycle presented in

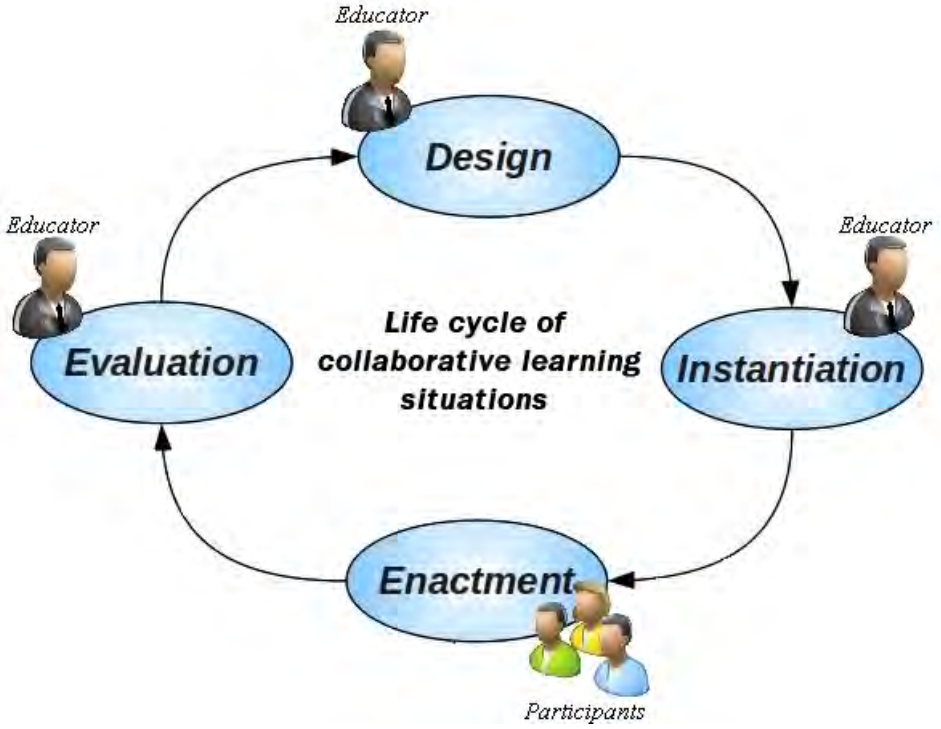

Figure 2.2: Life cycle of collaborative learning situations including the four phases: design, instantiation, enactment, and evaluation.

this section. Similarly, authors in [Cae06a, Per10] also consider the three first phases, which they denominate design-time, instantiation-time and run-time. A somewhat different example is the proposal of Dalziel [Dal06], who partially merges the instantiation phase in the design and enactment phases. Dalziel's life cycle and the adaptation to the one presented in this section is later exemplified in this document, when discussing the implementation of Dalziel's life cycle in the LAMS. As a conclusion, the life cycle presented in this section is consistent with other equivalent life cycles in the literature, which are accepted by the learning design and the CSCL communities.

Finally, it is convenient to clarify that this dissertation mainly contributes to the *instantiation* and *enactment* phases of the life cycle presented here. First, it contributes to the instantiation, since educators select and configure the specific tools that support the individual and collaborative activities in this phase. Second, it contributes to the enactment, because it is in this phase where students use these tools to accomplish the activities and achieve the learning objectives. It is important to note that this dissertation does not affect the way educators design the learning activities, since no new tool abstractions [Veg08] are defined. Nevertheless, the particularization of this life cycle with examples of VLEs and tools is later studied in this document, and remarkably, the design, instantiation and enactment phases may merge in some cases.

2.3. Virtual Learning Environments

Virtual Learning Environment (VLE) is a popular term in education, though a common definition for it has not been agreed yet, and many times other terms like Learning Management System (LMS), Content Management System (CMS), or Learning Platform (LP), are used as analogous concepts. One of the first and most popular definitions for VLE was provided in 2000 by the Joint Information Systems Committee (JISC) in the UK, and refers to "the components in which learners and tutors participate in online interactions of various kinds, including online learning"[1]. Nevertheless, many authors disagree on the need for online interactions [Dil02b], excluding this part of their own definition. This is the case of Stiles, who stated that both VLEs and LMSs are "designed to act as a focus for students' learning activities and their management and facilitation, along with the provision of content and resources required to help make the activities successful" [Sti00]. Similarly, Britain and Liber also excluded the online communication from their definition, while adding the collaboration as a remarkable feature, thus characterizing VLEs as CSCL systems: VLEs "aim to accommodate a wider range of learning styles and goals, to encourage collaborative and resource-based learning and to allow greater sharing and reuse of resources" [Bri06].

One of the most formal and commonly cited definitions of VLEs comes from Dillenbourg's work [Dil02b] published in 2002, and argued upon seven distinctive features.

1. VLEs are designed information spaces in which multiple authors can produce both structured and unstructured information.

2. VLEs are social spaces that promote interactions and discussions both synchronously or asynchronously.

3. VLEs are explicitly represented ranging from text-based interfaces to complex 3D graphical systems.

4. VLE students are also actors, thus producing, rather than consuming contents.

5. VLEs are not restricted to distance education, supporting also presential and blended learning situations.

6. VLEs integrate heterogenous technologies (including a variety of tools supporting different tasks), and multiple pedagogical approaches, because integration is under the idea of environment.

[1] `http://www.jiscinfonet.ac.uk/InfoKits/effective-use-of-VLEs/intro-to-VLEs/introtovle-intro`. Last visited: June 2012.

7. Most VLEs overlap with physical environments at some point, since learning activities may involve the use of non-computerized resources or interactions among participants.

Besides, it is noteworthy that Dillenbourg also stressed the affordances of VLEs for collaborative learning in his definition, due to the opportunities they offer for: "structuring collaboration", through the specification of learning scenarios or scripts; and "regulating interactions", through the monitoring of educators to ensure all group members participate in the activities.

From all the features reported in Dillenbourg's definition of VLEs, two of them are the basis for the research developed under this dissertation: the *integration of heterogeneous technologies*, and the *support of collaboration*. Therefore, despite consistent with Dillenbourg's and other accepted definitions in literature, this dissertation offers a broader characterization for VLEs, highlighting the aforementioned two features, and referring to *educator-centered*[2] *systems that allow the design, instantiation, enactment, and evaluation of collaborative learning situations through a set of synchronous/asynchronous, face-to-face/distance, individual/collaborative learning activities, which are supported by a collection of available tools and resources.* This definition eases the discussion regarding the proper use of terms like VLE, LMS or CMS, since all of them, as well as many of the LP, may fit under this description, as it is further discussed.

The remainder of this section presents multiple examples of VLEs, emphasizing those with a higher adoption among practitioners and institutions. Besides, the generic life cycle of collaborative learning situations, which was introduced in the last section, is particularized for these outstanding VLEs.

2.3.1. Examples of VLEs

Moodle, LAMS, .LRN, Sakai, Blackboard, Claroline or SharePoint LMS are some of the most widespread VLEs worlwide. Nevertheless, many others examples with a lower adoption are available, some of which might be trending VLEs in the next few years. Remarkable examples of these other VLEs are: Dokeos[3], Desire2Learn (D2L)[4], JoomlaLMS[5], RCampus[6], ILIAS[7], ATutor[8], eCollege[9] or Alphastudy[10]. Besides, it is noteworthy that a few other VLEs achieved a certain degree of success during the last decade; this is the case of WebCT, which now belongs

[2] in the sense that educators determine the activities, resources and tools in the learning design, as opposed to Personal Learning Environments (see "Other platforms used in education" in section 2.3.1)

[3] `http://dokeos.com`. Last visited: June 2012.

[4] `http://desire2learn.com`. Last visited: June 2012.

[5] `http://joomlalms.com`. Last visited: June 2012.

[6] `http://RCampus.com`. Last visited: June 2012.

[7] `http://ilias.de`. Last visited: June 2012.

[8] `http://atutor.ca`. Last visited: June 2012.

[9] `http://ecollege.com`. Last visited: June 2012.

[10] `http://alphastudy.com`. Last visited: June 2012.

to Blackboard. Finally, there is another important group of VLEs that were the results of research works and never became stable systems, like Gridcole [Bot08] or Pelican [Vel09], but that significantly contributed to the community working on this field.

The seven most widespread VLEs are studied next from a technical and functional perspectives emphasizing the features that enable the communication with external systems, as well as the functionality to support collaboration and groupwork. Table 2.1 summarizes and compares the main features analyzed for these VLEs along this section. Remarkably, while they are all based on web technologies, follow a classical three-tier client-server architecture [Eck95], offer extension APIs (Application Programming Interfaces) and include enough functionality for the support of collaboration, groupwork and role distinction, they are very heterogeneous on their pedagogical approaches and programming languages.

Table 2.1: Feature analysis of the main VLEs.

Feature	***Moodle***	***LAMS***	***.LRN***	***Sakai***	***Blackboard***	***Claroline***	***SharePoint LMS***
Current stable version (June 2012)	Moodle 2.2	LAMS 2.4	.LRN 2.5.0	Sakai 2.8.0	Blackboard Learn 9.1	Claroline 1.10	SharePoint LMS 3.1
Business model	Free	Free	Free	Free	Commercial	Free	Commercial
Technical features							
Distribution	Open source	Open source	Open source	Open source	Proprietary	Open source	Proprietary
System	Web-based system	Web-based system	Web-based system	Web-based system	Web-based system	Web-based system	Web-based system
Architecture	Three-tier client-server	Three-tier client-server	Three-tier client-server	Three-tier client-server	Three-tier client-server	Three-tier client-server	Three-tier client-server
Programming Language	PHP	Java	TCL	Java	Java	PHP	.NET
Extension API	✔	✔	✔	✔	✔ (private)	✔	✔ (private)
Functional features							
Pedagogical approach	Course-based learning	Activity-based learning	Community and course-based learning	Course-based learning	Course-based learning	Course-based learning	Course-based learning
Support of scripting	✘	✔	✘	✘	✘	✔	✔
Support of collaboration	✔	✔	✔	✔	✔	✔	✔
Support of groups	✔	✔	✔	✔	✔	✔	✔
Support of roles	✔	✔	✔	✔	✔	✔	✔
Number of built-in tools	14	25	15	20	16	8	16

Moodle

Moodle is currently the most successful VLE, with more than 66,000 registered sites and almost 60 million users all over the world at the time of writing[11]. Its claimed grounding on social constructionist [Jon99], its great flexibility regarding the management of learning activities, and of course, its free and open source distribution have fostered its quick adoption among institutions and practitioners worldwide [Mar08b]. Besides, its simple implementation using mature web technologies, its easier extension mechanisms compared to other VLEs, and the existence of plenty of development documentation have also promoted the creation of a huge community of people that contributes to the improvement of Moodle. This community is constantly developing new features and applications through hundreds of Moodle plugins[12], as well as supporting and advising other developers and end-users. Moodle first version (1.0) was released in 2002, as part of Martin Dougiamas' research on the use of open source software for teaching and learning within Internet-based communities [Dou99, Dou00]. Nonetheless, a lot of work has been done since this first version, and Moodle is currently distributed in the 2.2 stable version, being the 2.3 version announced for June 2012.

Moodle pedagogical approach is centered in the concept of course, which contains a set of activities and resources [Dou03]. Courses can be arranged in three different formats: weekly (the course is split in weeks, scheduling activities and resources along the time), topics (the course is divided into topics, each of which is composed by a structure of activities and resources), and social (the course is centered around a collaborative forum and its discussions). The concept of activity in Moodle is closely related to that of a tool, and 14 built-in activities are currently included in the Moodle distribution (see Figure 2.3), each of them mapped to a different kind of tool. Some examples of Moodle built-in tools are: forum, chat, quiz, multiple choice and wiki. Nevertheless, Moodle administrators can install a wide range of additional plugins to extend the set of activities of a Moodle installation.

Moodle promotes collaboration to engage students in the learning process [Cor05], enabling educators the creation and management of groups. These groups are defined in two different levels (groups and groupings). Those students belonging to a certain group or grouping share the same resources in each activity. Nevertheless, Moodle group configurations also allow the visualization of the rest of peers' resources, by setting the "visible groups" option. It is noteworthy that the design of a Moodle course follows a bricolage philosophy [Ber05], and so, educators can refine and iterate on the learning design as the course is being delivered, rather than completing it before the realization of the activities. This design philosophy affords a high flexibility, and educators can easily react to common occurrences in the classroom, such as no-shows or latecomers, modifying on-the-fly the structure of the course or the composition of the groups. This flexibility however,

[11] `http://moodle.org/stats`. Last visited: June 2012.
[12] `http://moodle.org/plugins`. Last visited: June 2012.

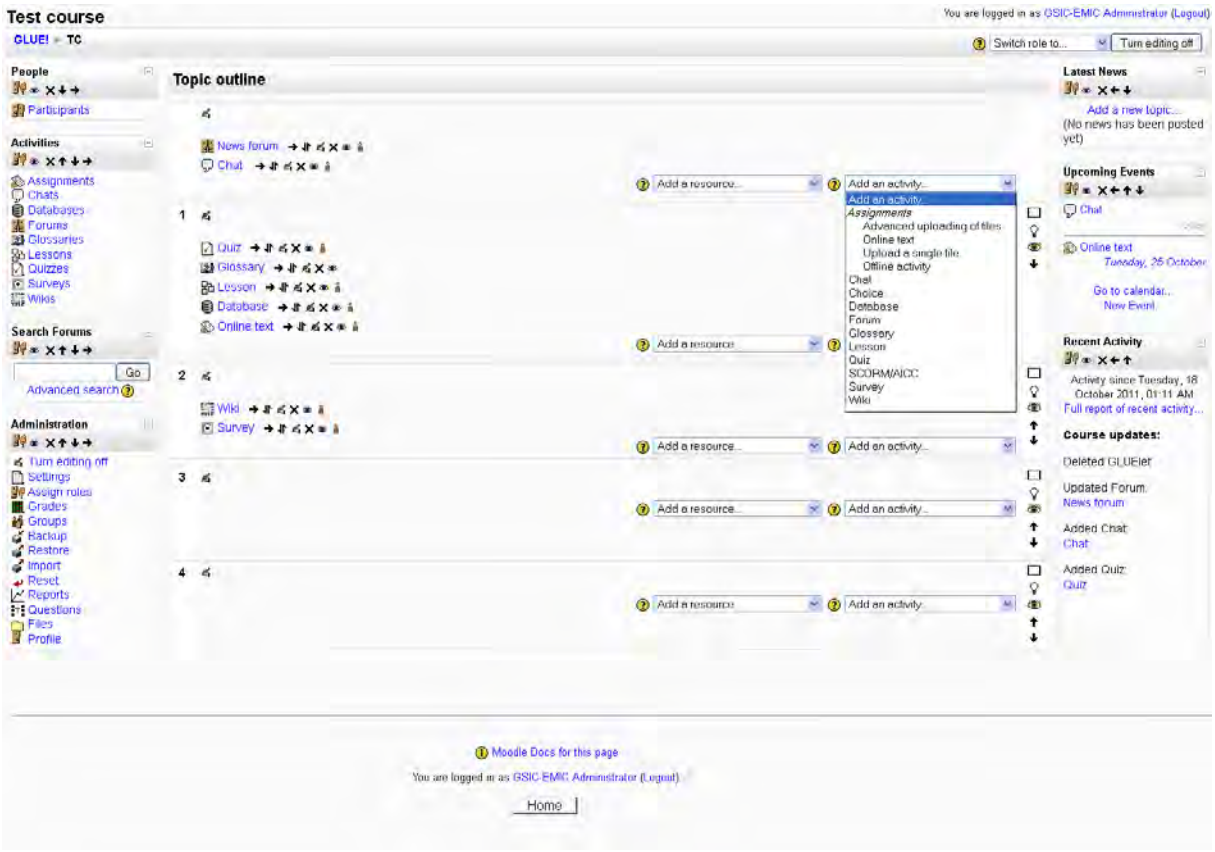

Figure 2.3: Moodle screenshot showing the structure of a course.

has an important limitation: it hinders the enactment of those designs that require the realization of the learning activities in a certain order, following a more sequenced and predefined scripted approach.

Regarding the technical side, Moodle is developed in PHP language and, as most VLEs, uses a common three-tier client-server architecture [Eck95]. The presentation tier defines how the information is presented to end-users, who access to that information using their web browsers. Business logic tier handles the interactions with Moodle, accessing to the data tier, typically through MySQL, to store and retrieve information when needed. Moodle is distributed in a single multiplatform package that includes PHP, MySQL and the Apache HTTP Server to facilitate the installation and configuration processes. Significantly, the current Moodle distribution complies with several educational standards. For instance, Moodle can manage ADL SCORM [ADL04] packages containing isolated resources or complete courses. Regarding IMS standards, Moodle supports IMS Content Packaging (IMS CP) [IMS07], IMS Common Cartridge (IMS CC) [IMS11], IMS Question and Test Interoperability (IMS QTI) [IMS06b], and is listed as one of the IMS Basic LTI [IMS10b] compliant VLEs.

LAMS

LAMS is a well-known free and open source web-based VLE for designing and enacting learning activities, first released in 2003 [Dal03]. Unlike Moodle and other VLEs, LAMS was explicitly developed with the purpose of facilitating the creation and management of learning designs [Dal05]. To do so, LAMS includes an intuitive and visual drag-and-drop interface where learning designs can be generated. Besides, these learning designs can be exported and shared among educators by means of LAMS. Actually, LAMS design philosophy focuses on the concept of individual activities, which may be sequenced as small lessons [Dal10], rather than as complete courses, like it happens, for instance, in Moodle. These lessons are more likely to be reused by different educators in different contexts, especially if they are based on recurrent pedagogical structures. In this context, LAMS providers have recently released the Activity Planner[13], a set of templates for designing lessons based on good teaching practices that can be imported and edited by educators within a LAMS environment.

LAMS is supported by a big community of end-users, administrators and developers, called the LAMS community[14]. According to the official statistics[15], in April 2011 there were more than 6,700 members from 80 different countries in this community. Besides, more than 800 lessons could be previewed and downloaded, to be used as starting templates when designing new learning sequences. Significantly, the promotion of design reuse brings important consequences to the implementation of LAMS, as regards the management of activities, tools and resources, since this VLE advocates for completing sequenced lessons in the LAMS authoring environment, before their deployment and execution in the LAMS monitoring environment [Bow11]. Therefore, educators' burden can be split in two roles (author and monitor), which may be played by different users, as opposed to Moodle and other similar VLEs. This can be seen as a consequence of the Educational Modeling Languages (EMLs) [Vig06] influence in LAMS, especially IMS LD [IMS03]. Nevertheless, despite sharing some features, like the typical export format [Dal06], LAMS avoided being a reference implementation of any EML.

Similarly to other VLEs, the concept of activity in LAMS can be mapped to that of a tool. LAMS current 2.4 version provides 25 built-in tools (Figure 2.4), including chat, forum, spreadsheet, noticeboard, mindmap or wiki [Dal10]. Most of these tools were developed from scratch by the own providers, although external developers may also extend this VLE with new tools. To do so, they must follow the LAMS integration contract [Ghi06], which details the interfaces, data models and technologies of this VLE. The success of external contributions in LAMS is quite limited at the moment, with two exceptions: the integration of Google Maps[16]

[13] `http://lamsinternational.com/product/activityplanner.html`. Last visited: June 2012.

[14] `http://lamscommunity.org`. Last visited: June 2012.

[15] `http://lamscommunity.org/dotlrn/clubs/educationalcommunity/forums/message-view?message_id=1261220`. Last visited: June 2012.

[16] `http://maps.google.com`. Last visited: June 2012.

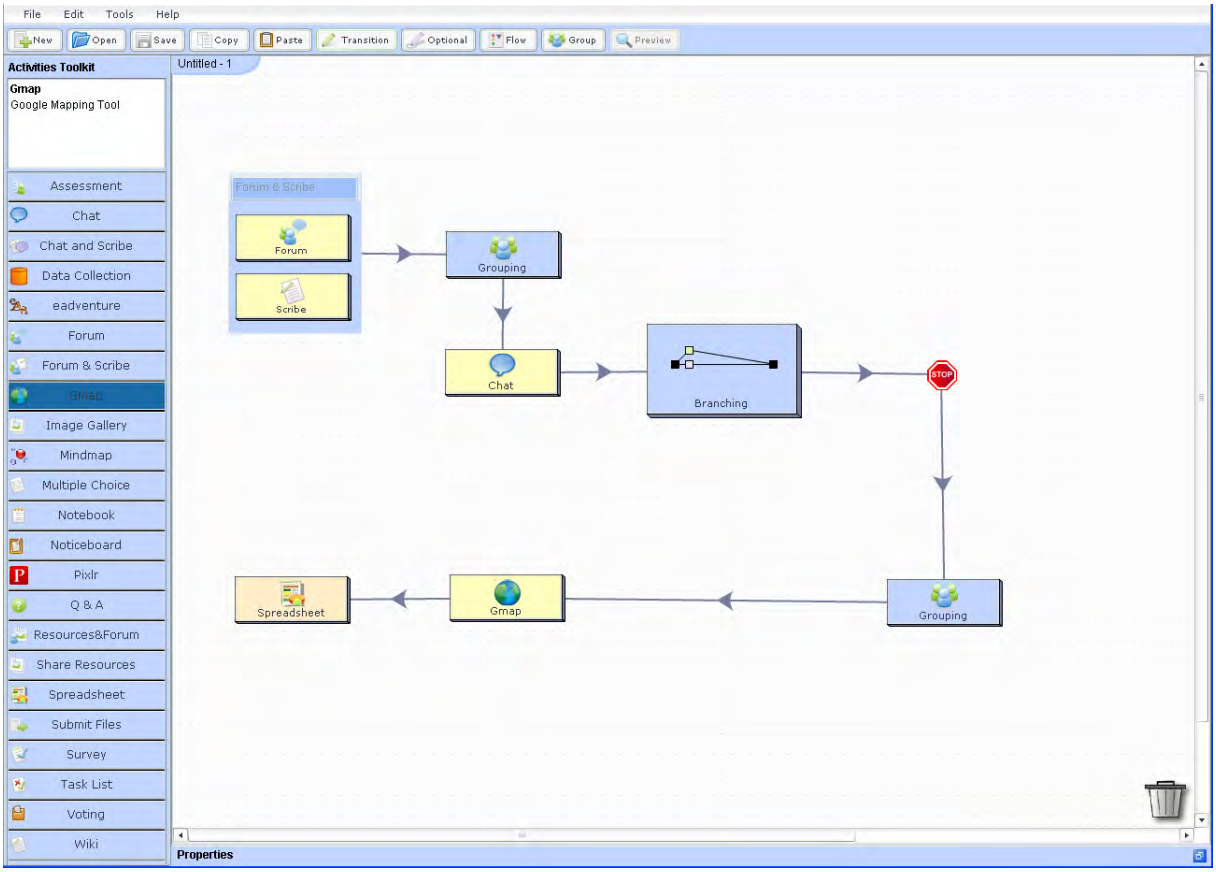

Figure 2.4: LAMS screenshot showing the design of a lesson in the authoring environment.

and <e-Adventure>[17]. In order to include LAMS tools as activities in a lesson, they must be dragged and dropped in the LAMS authoring environment, creating besides, linear learning flows by drawing lines between activities to indicate progression over time [Dal10]. Every LAMS activity can be configured to support groupwork and collaboration. Furthermore, branchings can be added to a lesson, so that every group may follow a different learning path, depending on educators or students' choice, or even on the outcomes of earlier activities. Once a lesson is designed, it can be deployed (instantiated) in the LAMS monitoring environment indicating the participants belonging to each group. Afterwards, students can enact the proposed activities in the LAMS learning environment.

Going into technological details, LAMS is developed in Java language, using the open-source Apache Struts[18] and Spring[19] frameworks, plus the Hibernate libraries[20] for Java. It

[17] `http://e-adventure.e-ucm.es`. Last visited: June 2012.
[18] `http://struts.apache.org`. Last visited: June 2012.
[19] `http://springsource.org`. Last visited: June 2012.
[20] `http://hibernate.org`. Last visited: June 2012.

also follows a three-tier client-server architecture with MySQL to access persisted data, and runs in a JBoss[21] application server. LAMS is normally distributed as a single multiplatform package, although an online website, called LessonLAMS[22], offering the equivalent authoring, monitoring and learning environments has become recently available. Finally, it is noteworthy that LAMS compliance with educational standards has always been very reduced, although in the later version (2.4) some efforts have been put to comply with IMS CC and Basic LTI.

.LRN

.LRN (or dotLRN) is a free and open-source web-based VLE that was originally developed at the Massachusetts Institute of Technology (MIT) using the OpenACS (Open Architecture Community System)[23] framework. .LRN is based on a rather user-centered approach [San07], as compared to other VLEs, although it allows the definition of several roles and hierarchies with different levels of permissions, similarly to what happens in Moodle or LAMS. Therefore, .LRN may be categorized halfway between educator-centered VLEs and student-centered PLEs. Its pedagogical approach enables the creation and management of courses and communities. The concept of .LRN course is equivalent to that in other VLEs. On the other side, communities are explicitly defined to foster interactions among .LRN users with special interests, projects, or just for social gatherings [Cal03]. Significantly, the concept of community is very common in education, especially at the university level, where students normally join to different clubs and societies.

Each .LRN course or community uses its own set of applications, templates and permissions [Cal03]. The current .LRN 2.5.0 version provides 15 built-in applications, like forum, webmail, news, calendar, file sharing or test, many of which support collaboration and groupwork in both course and community settings [San07]. Figure 2.5 shows a screenshot of the structure and tools of a community created within the .LRN environment. End-users can also create and manage learning units, which can be seen as lessons in the LAMS terminology, within their courses and communities. Nonetheless, these learning units must comply with the IMS LD specification, which is supported by .LRN through one of the most relevant extension packages.

On technical details, .LRN is developed in TCL [Ous89], a non-object oriented programming language, on top of the OpenACS framework, thus inheriting all its functionalities and features [Cal03]; actually, some .LRN tools, like the calendar or the webmail, are originally from OpenACS [Cal03]. .LRN also follows a three-tier client-server architecture, where data is managed through Postgresql or Oracle. An AOLserver[24] is employed to deliver .LRN content,

[21] `http://jboss.org`. Last visited: June 2012.
[22] `http://lessonlams.com`. Last visited: June 2012.
[23] `http://openacs.org`. Last visited: June 2012.
[24] `http://aolserver.com`. Last visited: June 2012.

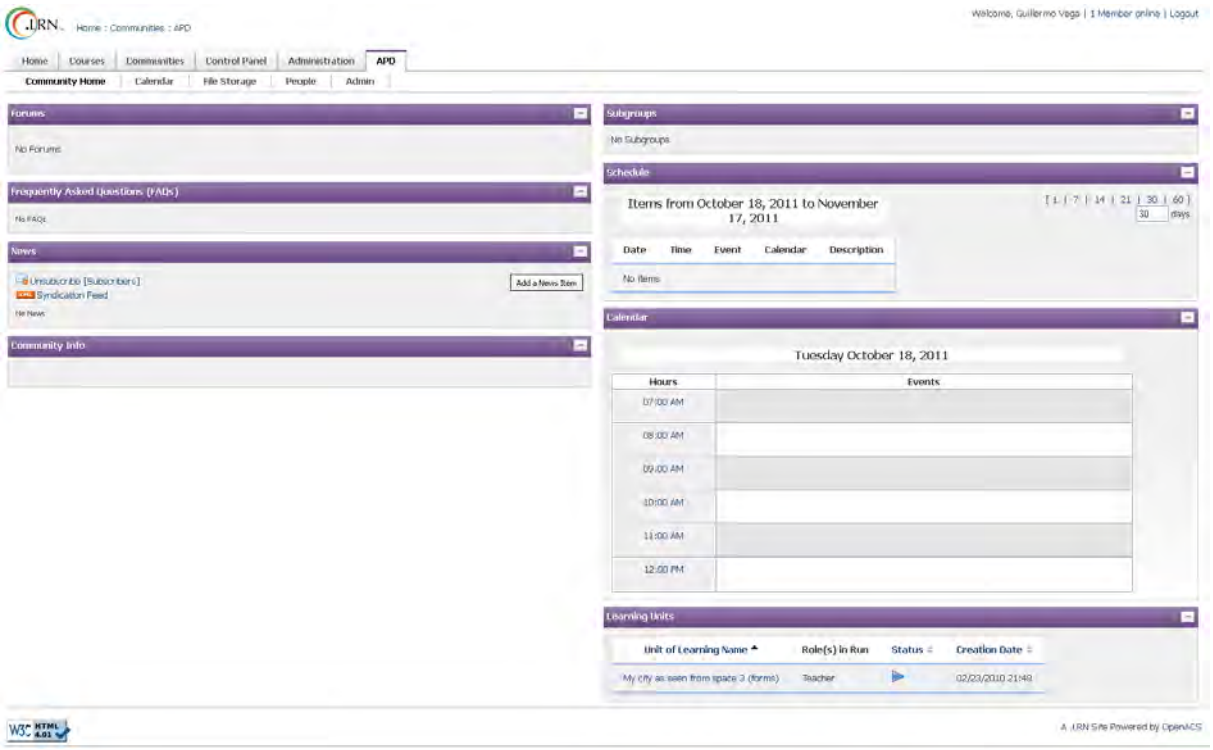

Figure 2.5: .LRN screenshot showing the structure of a community.

which can be accessed by end-users through their web browsers. Here, it is important to stress the high compliance of .LRN with many of the e-learning standards (apart from IMS LD) [San07], like IMS CP, IMS QTI, SCORM, or IMS Meta-data (IMS MD) [IMS06a].

.LRN had a certain impact short after its first released in 2002, being adopted as the *de facto* VLE in several institutions worldwide apart from the MIT, such as the University of Cambridge (UK), the University of Sydney (Australia), or the Universidad Nacional de Educación a Distancia (Spain) [Cal03]. Nevertheless, the popularity of .LRN has fallen sharply in the last few years, being quickly replaced by other VLEs like Moodle, LAMS or Blackboard with simpler programming languages and well-known learning structures. Actually, very few research works using .LRN have been reported during the last two years, and even the anual Open ACS / .LRN conference was canceled after 2009.

Sakai

The Sakai Collaboration and Learning Environment (Sakai CLE) or simply Sakai is a free and open-source web-based VLE designed and developed as part of the Sakai project[25]. This

[25] `http://sakaiproject.org`. Last visited: June 2012.

project was started in 2004 by five American institutions, releasing the first Sakai version in 2005. Although the popularity of Sakai cannot be compared to that of Moodle, over 350 institutions worldwide have adopted this VLE[26], and more than 70 partners are involved in the Sakai project, contributing with funds, or developing code [Sim07].

Sakai pedagogical approach is, like Moodle, based on the concept of course, in which educators add different unsequenced built-in tools and resources, in order to support students in the realization of their learning activities. Sakai 2.8.0 provides 20 "core tools", including announcement, chat, glossary, syllabus or tests & quizzes[27] (Figure 2.6). Most of these tools support collaboration, and their content can be shared among the participants belonging to the same group within a Sakai course. However, some other tools, like the email, despite fostering users' interactions, are not course-aware, being directly used in each user's workspace. Besides, some extra tools like Gradebook2[28], or the web conference tool Big Blue Button[29] can be installed and configured by Sakai administrators.

Sakai is developed in Java language, following a classical three-tier client-server architecture, in which the data layer is accessed through MySQL. It normally runs on Apache Tomcat servlet containers, although Apache HTTP servers or JBoss servers could also be used instead. Like most VLEs, Sakai is distributed in a single pre-built package that includes Apache Tomcat, thus facilitating the installation and configuration processes. Significantly, Sakai was designed to seek compliance with open e-learning standards [Far05], and many of them are currently supported, including SCORM, IMS CC, IMS CP, IMS LIS (IMS Learning Information Services) [IMS10a], Basic LTI or the W3C recommendation WCAG (Web Content Accessibility Guidelines) [W3C08].

Blackboard

Blackboard Learn or simply Blackboard is currently the most popular commercial VLE, with more than 1,000 clients using the latest 9.1 version[30], as well as the main product offered by the Blackboard Inc. company. The first Blackboard version was launched in 1998 gaining a quick adoption, especially among American Universities and learning centers; however, Blackboard users can nowadays be found all over the world. In 2006, Blackboard Inc. acquired WebCT, another proprietary VLE with a competitive presence at that time, continuing just the development of one single brand called Blackboard.

[26] `http://sakaiproject.org/adopt`. Last visited: June 2012.
[27] `http://sakaiproject.org/learning-management`. Last visited: June 2012.
[28] `http://confluence.sakaiproject.org/display/SG2X/Gradebook2`. Last visited: June 2012.
[29] `http://bigbluebutton.org`. Last visited: June 2012.
[30] `http://blackboard.com/about-bb/news-center/press-releases/Archive.aspx?releaseid=1481788`. Last visited: June 2012.

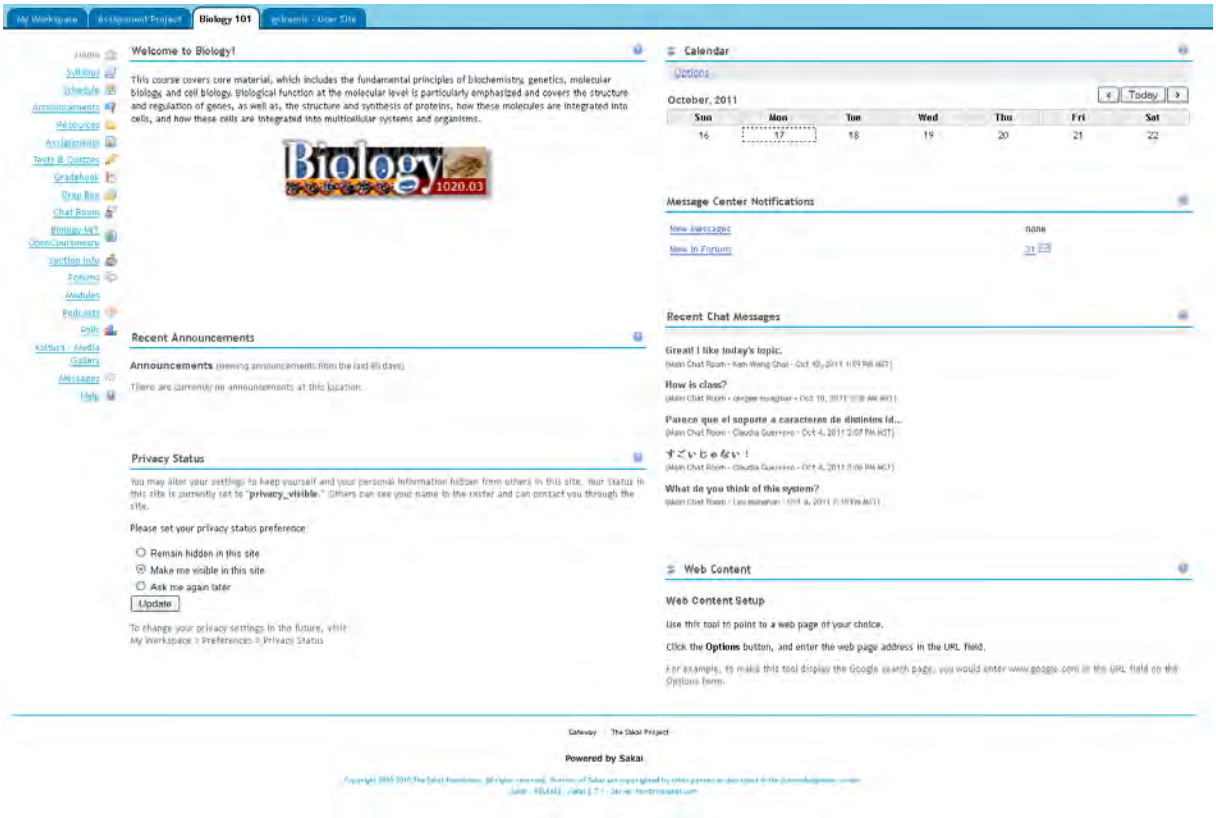

Figure 2.6: Sakai screenshot showing the structure of a course.

Blackboard pedagogical approach is, as that of Moodle or Sakai, based on the concept of courses. Educators can thus add resources and tools (named modules in the Blackboard terminology) within a Blackboard course, so that students can carry out the learning activities. Educators can also create and manage groups of students sharing the same resources or tool settings. The current Blackboard version provides 16 built-in tools, as it can be seen in Figure 2.7, some of which are announcements, blogs, journals, or tests. Furthermore, it includes additional features for collecting evidences and outcomes from students' work, as well as rubrics for formative assessment. Despite being a proprietary software, external developers have contributed to the extension of the functionality in Blackboard through the implementation of Building Blocks [Pit03]. Besides, Blackboard providers have developed some extra blocks to include some popular non-commercial external tools in recent versions, like the video hosting service Youtube[31], the slide hosting service Slideshare[32] or the social networking site Facebook[33] [Kat10].

[31] `http://youtube.com`. Last visited: June 2012.
[32] `http://slideshare.net`. Last visited: June 2012.
[33] `http://facebook.com`. Last visited: June 2012.

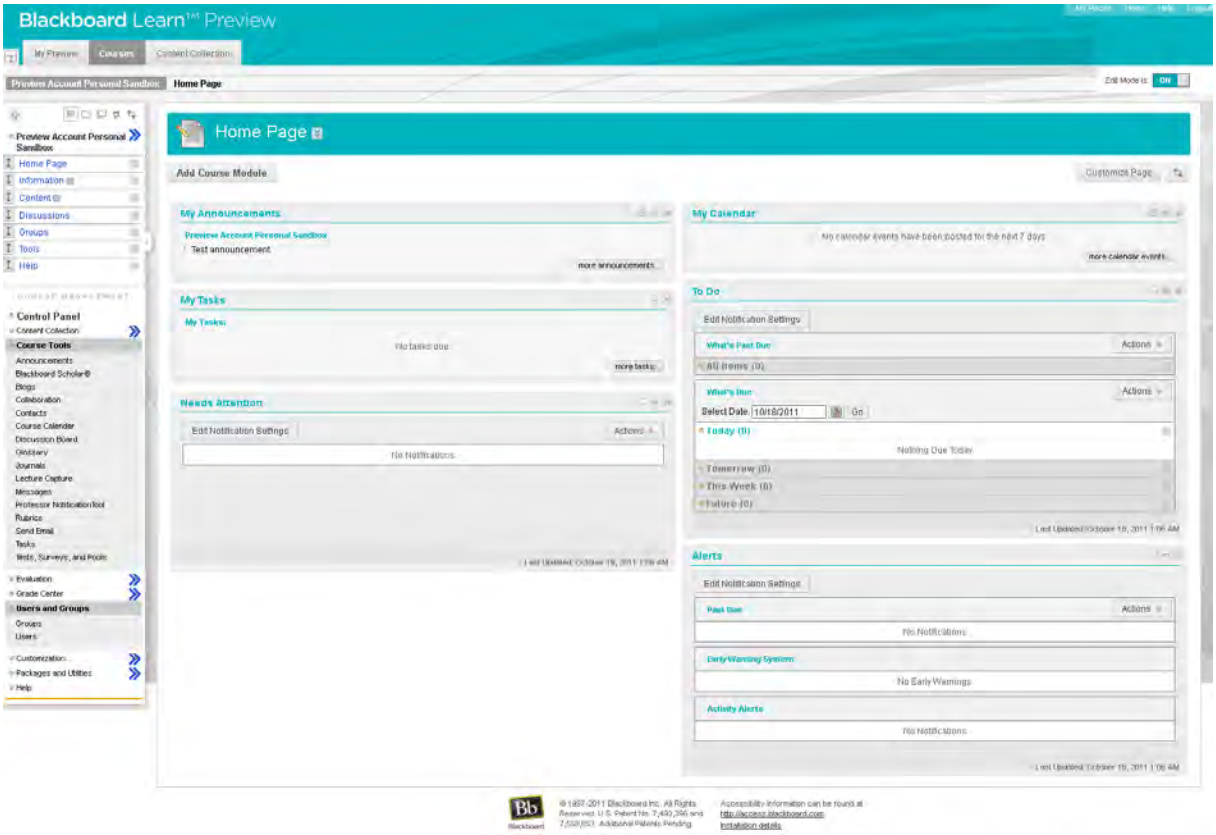

Figure 2.7: Blackboard screenshot showing the structure of a course.

Blackboard 9 is developed in Java, although previous versions included both Peal and Java. It follows a three-tier client-server architecture in which persistence is managed using Oracle or Microsoft SQL Server. Blackboard runs on Apache HTTP servers, enabling the access to end-users via web browsers. Regarding the support to e-learning standards, Blackboard is listed as a compliant partner for Basic LTI, IMS CC and SCORM.

Claroline

Claroline (Classroom Online) is a free and open-source VLE originally designed and developed by Thomas De Praetere [Wal04] in the Catholic University of Leuven, Belgium, and first released in 2001 [Leb09]. Official statistics report that almost 2,200 organizations in 113 countries (especially in European countries like Greece, France, Italy or Belgium) are using Claroline[34]. Besides, an increasing research interest on this VLE should be noticed, due to the important number of recent publications related to Claroline [Liu10], which also has its own annual international conference for developers and practitioners.

[34] `http://claroline.net/worldwide`. Last visited: June 2012.

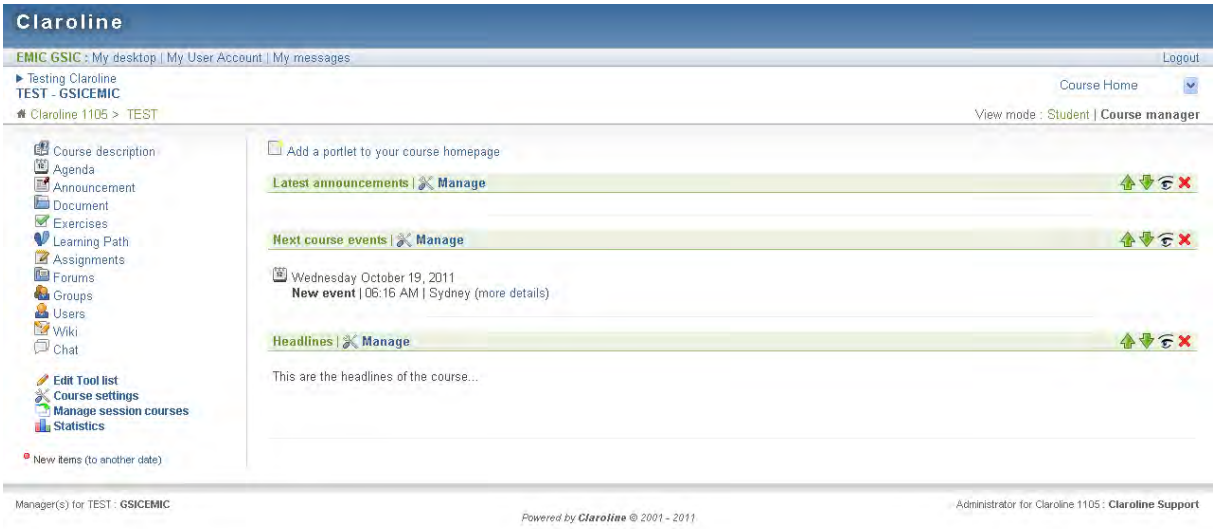

Figure 2.8: Claroline screenshot showing the structure of a course.

The pedagogical approach in Claroline is, like in Moodle or Sakai, based on courses, which are created and administered by teachers[35]. Teachers can create and manage groups within Claroline courses to promote collaboration, which is a significant feature explicitly reported in the pedagogical principles of Claroline. Besides, learning paths defining sequences of activities can be added to Claroline courses, forcing students to follow a set of steps when enacting collaborative leaning situations; this is analogous to LAMS lessons. Practitioners using the current 1.10 Claroline version are however very limited by the reduced set of 8 built-in tools that can be included in Claroline courses (and also in learning paths), as it can be seen in Figure 2.8. These tools are: agenda, announcement, document, exercises, assignments, forum, wiki and chat. Significantly, very few modules have been developed to extend Claroline with new tools, being most of them, like the web conferencing DimDim[36], incompatible with recent Claroline versions[37].

Regarding technological features, Claroline is developed in PHP and follows a three-tier client-server architecture in which the data layer is accessed through MySQL. Claroline normally runs on Apache HTTP servers, allowing end-users to connect to this VLE employing web browsers. Unlike Moodle or Sakai, the Claroline installation package does not include Apache HTTP server, nor MySQL, hindering the installation and configuration processes. Finally, the compliance of Claroline with other educational standards is very poor and only SCORM and IMS QTI are announced.

[35] `http://doc.claroline.net`. Last visited: June 2012.
[36] `http://dimdim.com`. Last visited: June 2012.
[37] `http://w2.claroline.net/index.php`. Last visited: June 2012.

SharePoint LMS

SharePoint LMS is a commercial VLE that was originally developed in 2007 by the Danish company ElearningForce[38]. Sharepoint LMS is based on the proprietary Microsoft Office SharePoint Server[39]. There does not exist official data about the number of end-users or customers, but its vendors claim that "many companies, organizations and educational institutions worldwide are using SharePoint LMS for training across many different kinds of industries"[40], including universities, healthcare, manufacturing or financial and legal services.

SharePoint LMS follows the same course-based pedagogical approach that Moodle, Sakai, Blackboard or Claroline [Ele11], enabling educators the creation and management of Sharepoint LMS courses. These courses can be categorized according to different levels defined by an institution or organization. Learning paths including sequences of activities in a course can also be created within this VLE, as in the cases of LAMS or Claroline. SharePoint LMS promotes the collaboration and the communication among the participants in a course through the definition of groups and the use of chats, mailbox, discussion boards, or the Microsoft Live Meeting online conference tool. The current 3.1 version of SharePoint LMS includes a total of 16 tools, as it is shown in Figure 2.9. Some of these tools are also proprietary and developed from scratch for this VLE, like the Podcasting kit for SharePoint, or are offered by some external partners through special agreements, like the Plagiarism prevention tool [Ele11].

SharePoint LMS is developed in .NET, using the ASP.NET[41] framework from Microsoft, and can be extended by developing *web parts*, which are the equivalent to plugins in the ASP.NET terminology[42]. This VLE also follows a three-tier client-server architecture, running on a Microsoft Office SharePoint Server and using a Microsoft SQL server for persistence. Regarding compliance with e-learning standards, only SCORM and IMS QTI are supported.

Other platforms used in education

Despite the great success of VLEs, other platforms have also been commonly employed in educational scenarios in the last few years. Some outstanding examples of these platforms are: wikis [Leu01], social networking sites [Cho07], and PLEs [Har06]. These platforms share important technical features with the main VLEs, e.g. they are typically free and open source web-based systems that follow three-tier client-server architectures. Nevertheless, their functionality with respect to the management of activities, groups, roles and tools is limited, and in

[38] `http://elearningforce.dk`. Last visited: June 2012.
[39] `http://sharepoint.microsoft.com`. Last visited: June 2012.
[40] `http://sharepointlms.com/clients`. Last visited: June 2012.
[41] `http://asp.net`. Last visited: June 2012.
[42] `http://msdn.microsoft.com/en-us/library/e0s9t4ck.aspx`. Last visited: June 2012.

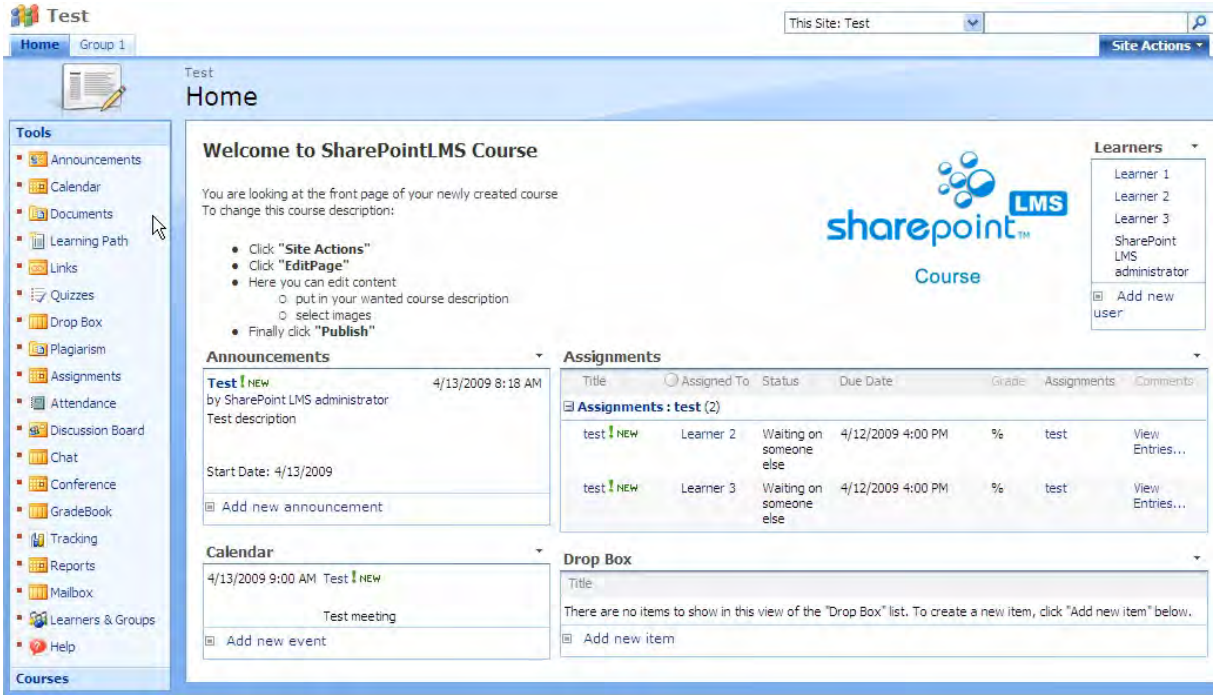

Figure 2.9: SharePoint LMS screenshot showing the structure of a course.

some cases nonexistent. Table 2.2 compares the same features analyzed in VLEs for representative examples of each of these platforms: MediaWiki (wiki), Facebook (social networking site), and the Southampton Learning Environment (PLE). None of these platforms can be considered a VLE, since they are not educator-centered systems; however, they can support the integration of external applications, as well as the design, instantiation, enactment and evaluation of collaborative learning situations, as discussed next, and so, they also appear in this analysis.

- **Wikis** are software systems that allow the creation and management of content structured in interlinked web pages [Leu01]. This content is normally edited by multiple users working in collaboration, and may include links to external resources. Some well-known examples of wiki websites are Wikipedia[43] (the online encyclopedia), Wiktionary[44] (the online dictionary) or Wikiversity[45] (the site for creating and sharing tutorials and learning materials). Most wiki sites run on the PHP-based MediaWiki[46] software application, which has also been used as a Content Management System in several organizations[47] and institutions. For example, authors in [Ala11a, Jor07, Mar08a] illustrate with real examples the use of

[43] `http://wikipedia.org`. Last visited: June 2012.
[44] `http://wiktionary.org`. Last visited: June 2012.
[45] `http://wikiversity.org`. Last visited: June 2012.
[46] `http://mediawiki.org`. Last visited: June 2012.
[47] `http://mediawiki.org/wiki/Sites_using_MediaWiki/corporate`. Last visited: June 2012.

Table 2.2: Feature analysis of other commonly employed platforms in education.

Feature	***MediaWiki***	***Facebook***	***SLE***
Current stable version (June 2012)	MediaWiki 1.19	hosted by the provider	Under development (expected during 2012)
Business model	Free	Free	Free
Technical features			
Distribution	Open source	Proprietary	Open source
System	Web-based system	Web-based system	Web-based system
Architecture	Three-tier client-server	Three-tier client-server	Three-tier client-server
Programming Language	PHP	PHP	.NET
External API	✔	✔	Under development
Functional features			
Pedagogical approach	-	-	Student-centered approach
Support of scripting	✘	✘	✘
Support of collaboration	✔	✔	✔
Support of groups	✘	✔	✔
Support of roles	✔	✘	✘
Number of built-in tools	2	6	Under development

MediaWiki in higher education courses to foster the collaboration among students. Nevertheless, MediaWiki functionality is quite limited for this purpose due to several reasons. First, MediaWiki has not been designed to follow any pedagogical approach. Besides, it does not support the concept of course, nor the concept of activity, nor the creation of groups (there is an extension for group-based access control but its use is not advisable[48]). Furthermore, very few built-in tools are included in the MediaWiki distribution (let them be upload file and collaborative text editor). Therefore, MediaWiki cannot be categorized as a VLE, even though it has shown to be useful for the design and enactment of some collaborative learning situations.

- **Social Networking sites** are online systems where users can create their own profiles and build a personal network that connects them to other users [Len07]. These sites have quickly proliferated in the last ten years, reaching millions of users worldwide. The most outstanding social networking sites are Facebook, LinkedIn[49], MySpace[50] and Google+[51]. These sites are normally free web-based application that, unlike VLEs, do not need to be installed and administered, because they are hosted by their own providers. Their widespread adoption and their focus on social interactions have encouraged the use of

[48] `http://mediawiki.org/wiki/Extension:Group_Based_Access_Control`. Last visited: June 2012.
[49] `http://linkedin.com`. Last visited: June 2012.
[50] `http://myspace.com`. Last visited: June 2012.
[51] `https://plus.google.com`. Last visited: June 2012.

these sites by educators in order to support the communication and collaboration among students in their courses [Loc08, Mad09, San11]. These social networking sites normally allow the creation and management of groups within the user's profile, but they are not designed to follow any pedagogical approach. Therefore, most educational features, such as the creation of activities, the formalization of learning objectives, or the role distinction, are not supported. Besides, social networking sites present very few built-in tools. In the case of Facebook, only chat, message, events, news, upload pictures, and a personal wall are included. Nonetheless, many Facebook extensions from external developers are currently available. In short, social networking sites cannot be categorized as VLEs, although they may be useful for the design and enactment of some collaborative learning situations.

- **Personal Learning Environments (PLEs)** are defined as software systems "that help students take control of and manage their own learning" [Har06]. PLEs are characterized by directly involving learners in the access, aggregation, configuration and manipulation of lightweight tools and resources (unlike VLEs in which educators select and manage the resources and tools that students must use) [Sev08]. PLEs also provide personal spaces and social contexts for collaboration [Cha07]. Actually, research on PLEs is a very promising field on TEL and CSCL for the next years, although only a few systems with a very low adoption (compared to that of VLEs), have been so far developed. Some examples of real PLEs that can be found in the literature are the prototype developed by Scott Wilson in the PLEX project [Wil06], or the Southampton Learning Environment (SLE) [Whi11], a customized PLE for the staff and students belonging to the University of Southampton (UK) that is expected to be ready during 2012 [Mil11]. Significantly, the SLE may be the first PLE with an institutional adoption after five years of research on this field, and so it deserves a further analysis. The SLE is being developed in .NET, running on Microsoft Sharepoint server 2010, and offering a free web-based access. SLE end-users are intended to have their own homepage that eventually becomes a canvas on which tools and resources can be dropped. These tools are planned to be developed as *web parts*, following the *App Store* model extension mechanism for mobile devices, although a yet undefined number of built-in tools, such as document spaces, wikis or forums, might be available in the final release [Mil11]. Besides, individual end-users could create and manage groups in the SLE, thus automatically granting access to these resources and tools to other group members. All in all, PLEs like the SLE rely on a non-hierarchical approach that promotes students' self-arrangement during their learning process, and even though they share some technical and functional features with VLEs, they cannot be categorized as such. However, due to the promotion of collaboration and groupwork, PLEs may also be very useful for the design and enactment of some collaborative learning situations.

2.3.2. Life cycle of VLEs in collaborative learning situations

As previously discussed, VLEs support the creation and management of groups, the distinction of roles, and include some built-in tools that promote the collaboration and communication among learners. Besides, in general, group configurations and role assignments may change as activities, lessons or courses, do. Therefore, VLEs can support the life cycle of collaborative learning situations presented in section 2.2.1, although the implementation of each of the four phases in this life cycle (i.e. design, instantiation, enactment and evaluation) varies from one VLE to another, mainly due to their different pedagogical approaches.

For example, in course-based VLEs like Moodle, Sakai, Blackboard, Claroline or Sharepoint LMS, educators can continuously design and instantiate new activities through the VLE graphical interface, while some other activities are being enacted and even evaluated. Thus, in this type of VLEs, the life cycle consists of many short iterations in the design, instantiation and enactment phases, while the evaluation can be carried out at each iteration or after finishing the enactment of the entire learning situation. This implementation of the life cycle is the consequence of a more flexible bricolage philosophy [Ber05], which enables educators to redesign as the course is being delivered.

Nevertheless, other VLEs like LAMS, which follows an scripting activity-based approach, promotes the completion of the learning design before the enactment and evaluation of the collaborative learning situation. Therefore, LAMS clearly splits design and enactment in two different phases, although some changes are allowed after the design is completed, but only before learners start the activities. Remarkably, in LAMS a part of the instantiation (tool particularization) is done during the design in the authoring environment, and another part (group population) before the enactment in the monitoring environment.

Authors like Bower [Bow11] have pointed out some important limitations educators find on VLEs in each of these four phases. In the design phase, most complains are due to the lack of pedagogical support and scripting when creating new learning designs. LAMS is one significant exception that provides ready-to-use templates for different teaching styles and a scripting-based design approach. In the instantiation phase, the main limitation is the reduced set of available tools that educators can select for their learning designs. This is a common and recurrent problem in all VLEs, as previously discussed. In the enactment phase, the problem is the limited set of tools learners can use, as a consequence of the same limitation found in the instantiation phase. Finally, in the evaluation phase, educators complain about the limited VLE functionality to process activity outcomes, and to get information about students' interactions. Here, it is convenient to remind the reader that the central scope of this dissertation is the limitation regarding the reduced set of available VLE built-in tools, which affects the instantiation and

enactment phases. Despite their research interest, limitations affecting the design and evaluation phases are beyond the scope of this research work.

Before concluding this section, a brief note can be pointed out on wikis, social networking sites and PLEs. These platforms are more frequently employed for informal settings, in which learning occurs with no (or few) support from educators. This peculiarity together with the lack of some collaborative features in these platforms, such as group configurations or role assignments, blurs the distinction of the four phases in the life cycle of collaborative learning situations; sometimes, even design, instantiation or evaluation phases may be completely missed.

2.4. Software tools

Tools, ranging from paper and pen to highly sophisticated interactive whiteboards have been traditionally employed to support human actions within education [Arm05]. The use of tools implemented as programming applications has played a very important role in the support of learning during the last thirty to forty years, mainly due to the fast development of computing, along with the incorporation of PCs (Personal Computers), and other electronic systems in our daily life [Rob97]. Nevertheless, it was not until the last decade, with the spreading of web technologies and the possibilities of a fast and ubiquitous access to information and resources, together with the quick proliferation of SaaS applications [Pap03], that these tools have been massively adopted at all stages, from formal to informal learning, from face-to-face to distance learning, and from primary school to higher education and beyond.

Furthermore, the current trend of users generating and sharing content through the so-called Web 2.0 tools [ORe07,Sol07], most of which are delivered as SaaS application, has changed the way learning happens [Ajj08], encouraging communication and collaboration and fostering a more constructivist learning. Some examples of these Web 2.0 and SaaS tools with an important adoption in education are the mapping service Google Maps, the social bookmarking service Delicious[52], or the image hosting service Flickr[53].

Additional examples of software tools with a high adoption among practitioners are presented along this section. Besides, the generic life cycle of collaborative learning situations, which was introduced in section 2.2.1, is particularized for these software tools, and compared with that of VLE built-in tools.

[52] `http://delicious.com`. Last visited: June 2012.
[53] `http://flickr.com`. Last visited: June 2012.

2.4.1. Examples of software tools

Practitioners worldwide are currently employing thousands of different software tools for the support of their learning activities. The most recurrent and valued of these tools are listed by several organizations, such as The Centre for Learning & Performance Technologies, which annually gathers the most outstanding tools according to the opinions of learning professionals, in the Top 100 Tools for Learning[54] list. Another similar list of tools but categorized according to the tasks these tools are designed for (e.g. drawing tools, music tools, presentation tools, etc.) can be found in the Cool Tools for Schools[55] site. Deeply analyzing in this dissertation the whole number of tools presented in these two lists would be a very overwhelming and unnecessary task, and so, only the seven most valued tools by learning professionals in the first list have been selected for a further study (as an analogy to the case of VLEs in the previous section). According to the 2011 Top 100 Tools for Learning list these seven tools are Twitter[56], Youtube, Google Docs[57], Skype[58], Wordpress[59], Prezi[60] and Slideshare. It is noteworthy that Moodle, which is placed in the 7th position, has been excluded from this further study, since in this dissertation Moodle is considered a VLE rather than a tool, and so, it was analyzed in the previous section. This selection of seven tools covers multiple categories from the second list, including video tools, collaborative tools, blog tools, communication tools, presentation tools and slideshow tools.

Table 2.3 summarizes and compares the technical and functional features of these seven tools. As it can be seen, they are all *freely offered*, although some upgrades can be obtained by paying a fee in Prezi, Wordpress or SlideShare. Besides, they are all based on *web technologies* (except for the client-server Skype), enable their access to end-users via web browsers over the Internet, and are offered as SaaS application. In addition, they all can be employed to support *collaboration and sharing*. These three features written in italics are stressed in Solomon's characterization of Web 2.0 tools [Sol07] in comparison to traditional software applications. However, some exceptions can be easily found: for instance, many e-portfolios like rGrade[61] or EdCube[62] are not free, while many personal sites like TiddlyWiki[63], or very simple tools like the online dictionary Wordreference[64] are not collaborative. Regarding the latter, some recent works like TaKo [Mon12] are tackling the conversion of personal and single-user applications into transparent collaborative multi-user tools.

[54] `http://c4lpt.co.uk/top-tools/top-100-tools-for-learning-2011`. Last visited: June 2012.
[55] `http://cooltoolsforschools.wikispaces.com`. Last visited: June 2012.
[56] `http://twitter.com`. Last visited: June 2012.
[57] `http://docs.google.com`. Last visited: June 2012.
[58] `http://skype.com`. Last visited: June 2012.
[59] `http://wordpress.org`. Last visited: June 2012.
[60] `http://prezi.com`. Last visited: June 2012.
[61] `http://rgrade.com`. Last visited: June 2012.
[62] `http://edcube.com`. Last visited: June 2012.
[63] `http://tiddlywiki.org`. Last visited: June 2012.
[64] `http://wordreference.com`. Last visited: June 2012.

Table 2.3: Feature analysis of the main software tools for learning.

Feature	***Twitter***	***Youtube***	***Google Docs***	***Skype***	***Wordpress***	***Prezi***	***Slideshare***
Current stable version (June 2012)	hosted by the provider	hosted by the provider	hosted by the provider	Skype 5.9 (Windows); Skype 2.2 (Linux); Skype 5.7 (Mac OS)	WordPress 3.3.1	hosted by the provider	hosted by the provider
Business model	Free	Free	Free	Free	Free / commercial	Free / commercial	Free / commercial
Technical features							
Distribution	Open source	Proprietary	Proprietary	Proprietary	Open source	Open source	Proprietary
System	Web-based system	Web-based system	Web-based system	Client-Server sytem	Web-based system	Web-based system	Web-based system
Architecture	Three-tier client-server	Three-tier client-server	Three-tier client-server	Peer-to-Peer	Three-tier client-server	Three-tier client-server	Three-tier client-server
Programming Language	Ruby	JavaScript, Adobe Flash	Java	Delphi, C++	PHP	Adobe Flash, Adobe AIR	Ruby
Public API	✔	✔	✔	✔	✔	✘	✔
Functional features							
Purpose	microblog, social networking site	Video hosting service	Office suite	Audio / Video-conference, instant messaging	Blog, CMS	Presentation tool	Slide hosting service
Support of communication	✔	✘	✘	✔	✘	✘	✘
Support of collaboration	✔	✔	✔	✔	✔	✔	✔
Support of groups	✘	✘	✔	✔	✘	✘	✘
Support of roles	✔	✔	✔	✘	✔	✔	✔

Comparing technical features, it is important to note the high heterogeneity with respect to the programming languages and the current trend of providing external APIs for these tools in order to promote their integration in other sites and platforms. Significantly, five out of the seven (Twitter, Youtube, Google Docs, Wordpress and Slideshare) provide REST-like (based on the Representational State Transfer architectural style) [Fie00] interfaces. These are language-independent interfaces that offer a restricted and well-defined set of methods aimed at facilitating the communication of these tools with other systems; however the specific resources and data formats employed to accomplish this communication are specifically given by each vendor.

Comparing functional features, only Twitter and Skype are explicitly designed to support communication, while the definition of groups and roles normally depends on the permissions granted over a certain content. For instance, a spreadsheet in Google Docs can be shared among a set of users, thus making up a group, similarly to what happens in a multiconference with

Skype. Regarding roles, all these tools (but Skype) present at least, two of them: producer and consumer of content. As an example, in Prezi or Slideshare an author may edit or upload presentations, while the remaining users may visualize them.

Nevertheless, not all the tools that are currently used for learning are web-based SaaS tools. This is the case of Synergo [Avo04], a Java standalone collaborative mapping environment developed at the University of Patras (Greece), and still used nowadays. In this example, other web tools with a similar functionality, like Text2MindMap[65] or Dabbleboard[66] could replace Synergo. However, in some other cases, especially in those involving tools designed and developed for a very specific educational purpose, there is no web-based SaaS alternative yet. This happens, for example, with the Distributed Network Simulator Environment (DNSE) [Bot10], a Java application based on grid technology intended for the parameter sweep simulation of computer networks, or with the also Java and grid-based Benchmarking Tool [Ala09] developed for computer architecture students to benchmark multiple workloads in different systems. Here, it is convenient to point out that some educators and institutions are still reluctant to use SaaS tools, due to the loss of control derived from using third-party hosted software, which entails additional security and privacy concerns regarding who access and uses the data [Cay09]. Nonetheless, most educators are progressively adopting SaaS tools, as previously exemplified in this section.

Concluding, there is a huge range of software tools that are employed to support individual and collaborative learning activities. However, VLEs and other similar platforms in which educators normally design and deliver their courses and lessons only include a very restricted and rather *ad hoc* set of built-in tools. Educators strongly emphasize this VLE limitation [Bow11], demanding more alternatives for the support of their collaborative learning situations. In this context, the integration of existing external tools in existing VLEs emerged as a research line aimed at tackling this problem [Dag07, Liv08]. This line could take advantage of the current trend in the massive development and use of SaaS and Web 2.0 applications. At this point, three important clarifications must be done. First, the integration of existing external tools in existing VLEs is only one solution to overcome this limitation, and even though it is the most accepted in literature [Ala12a] (especially from a generic perspective) other alternatives are feasible (see section 2.5). Second, SaaS and Web 2.0 applications have been quickly adopted in education, but there exist other non-SaaS and non-Web 2.0 tools with different technical and functional behaviors and. These other tools may be interested to support some learning activities, and so, they should also be taken into account when tackling the integration of external tools in VLEs. Last but not least, since most of these tools have been designed and developed to support collaboration, then their influence in the life cycle of collaborative learning situations should be further discussed.

[65] `http://text2mindmap.com`. Last visited: June 2012.
[66] `http://dabbleboard.com`. Last visited: June 2012.

2.4.2. Life cycle of software tools in collaborative learning situations

The life cycle of collaborative learning situations comprises four phases (see section 2.2.1): design, instantiation, enactment and evaluation [Gom09]. Nevertheless, the management and utilization of the software tools that support students' learning only involves two of these phases. More specifically, tools are selected and particularized according to the learning objectives, the social configuration and the structure of activities in the *instantiation phase*. Besides, they are used by learners in order to accomplish the activities and achieve the learning objectives in the *enactment phase*. Significantly, the outcomes produced by students as a result of using some software tools in the enactment phase (e.g. a document generated by a group of students using Google Documents) can sometimes be the input of the evaluation phase. However, in this fourth phase, educators evaluate the knowledge and skills acquired by the students, and so, the importance is in the content of these outcomes, and not in the tools employed to generate that content. Thus, the evaluation phase is considered independent of the tools that support students' learning. The same happens in the design phase, where only the tasks that should be performed by the participants (and not the specific tools supporting these tasks) are defined. It should be also noted that when VLEs are the environments supporting collaboration, some of these four phases may be blurred or merged, as discussed in section 2.3.2 with specific examples.

Practitioners managing software tools in the instantiation phase (being this phase supported by VLEs or not) must take into account two distinctive collaborative features, which have already been discussed. First, they must consider the social configuration of the learning situation, which describes the participants, their roles and their group settings. They must also consider the environment supporting the learning situation, in which participants find their particularized structure of activities, tools and resources [Bak97].

The *social configuration* requires that a different **instance** of each software tool is assigned to each student in individual activities and to each occurrence of a group in collaborative activities [Her06a, Her07b]. The term instance in the context of learning software tools typically comprises a *resource*, and a *software client* used to access to the resource [Bot08, Per10].

The type of each *resource* is determined by the nature of the tool. Examples of resources are: a document, in a text editor like Google Documents; a canvas, in a drawing tool like Dabbleboard; a presentation, in a slide or presentation tool like Prezi or Slideshare; a video, in a broadcasting service like YouTube; a chat room or an established conversation, in one of the many chats available in the web or in a conversational tool like Skype; an article, in an encyclopedia like Wikipedia; a concept map, in mind map tools such as Text2MindMap or Synergo. Some of these tools are inherently collaborative and hence, the resource must be shared (e.g. though nothing is wrong technically, it does not make sense to have a chat room for an individual student alone), while other tools can be used in collaboration or individually (e.g. if a document is shared among

students in Google Documents they can perform collaborative editing, whereas if the document is accessed only by one student the writing activity will be performed individually). It should also be observed that the usage of these resources means sometimes to modify them (e.g. writing on a document, drawing on a canvas), while some other times students just consume these resources (e.g. watching a video, reading an article from the encyclopedia).

To modify or consume these resources, students need *software clients* that run in their systems. In a few cases these clients may exist in the form of standalone applications (e.g. the Skype client). Nevertheless, in many cases, these clients run within students' browser making use of a series of web technologies, including HTML [W3C99], JavaScript [NWG06] or AJAX [Gar05], among the most popular ones. In these cases, the software client is downloaded (using the browser) from the service hosted by the tool provider, and then this client retrieves a resource representation to be consumed by the student and, if applicable, upload changes on that resource.

Given these premises, an instance of a learning tool is defined in this dissertation as *a resource that can be accessed and (if applicable) manipulated by the student, and the client software needed for it.* Examples of such instances are: a document in Google Documents plus the JavaScript code that enables end-users to see and edit the document in the browser; a video on YouTube plus the Flash (or HTML5 in the latter versions) code running on the browser that enables end-users to watch the video; or a Prezi presentation and the Flash code that enables viewing or editing the presentation in the browser. Significantly, non-web tools also conform to this definition; for example, in Synergo, an instance would be a concept map and the Java client of the standalone Synergo application.

In collaborative learning, the term instance gives the name to one of the four phases in the life cycle of collaborative learning situations: the instantiation phase. Therefore, in the instantiation phase, different tool instances need to be created and assigned to each user or group, for each activity. Interestingly, main VLEs like Moodle, LAMS, Sakai or Blackboard create different instances of their built-in tools and automatically assign them to each user or group defined in each activity of the collaborative learning situation.

On the other side, the *environment*, which is the other collaborative feature, requires a proper customization for each participant, regarding the structure of activities and tools, and depending on the social settings and the objectives of the collaborative learning situation. For instance, when applying a collaborative jigsaw pattern [Her11], each group strives to solve a small part of a bigger problem in the same activity, but normally using similar tools and artifacts. Therefore, these tools (and their instances) should be customized in the instantiation phase according to the particular learning objectives of each group, thus requiring a prior configuration before they are used by the participants [Per10]. The original idea of customizing tools before the realization of a learning activity comes from EMLs, like IMS LD [IMS03], and scripting [Mia05]. Most VLEs also support a prior configuration of their built-in tools.

Once tool instances are created and configured, they can be used by learners to support collaboration and groupwork in the enactment phase. Afterwards, once the collaborative learning situation is finished, educators may decide to delete these instances or to store their content during a certain period of time. Following these ideas, a life cycle for the management and utilization of tool instances within collaborative learning situations can be depicted. This tool life cycle can be drawn together with the aforementioned life cycle of collaborative learning situations, as represented in Figure 2.10

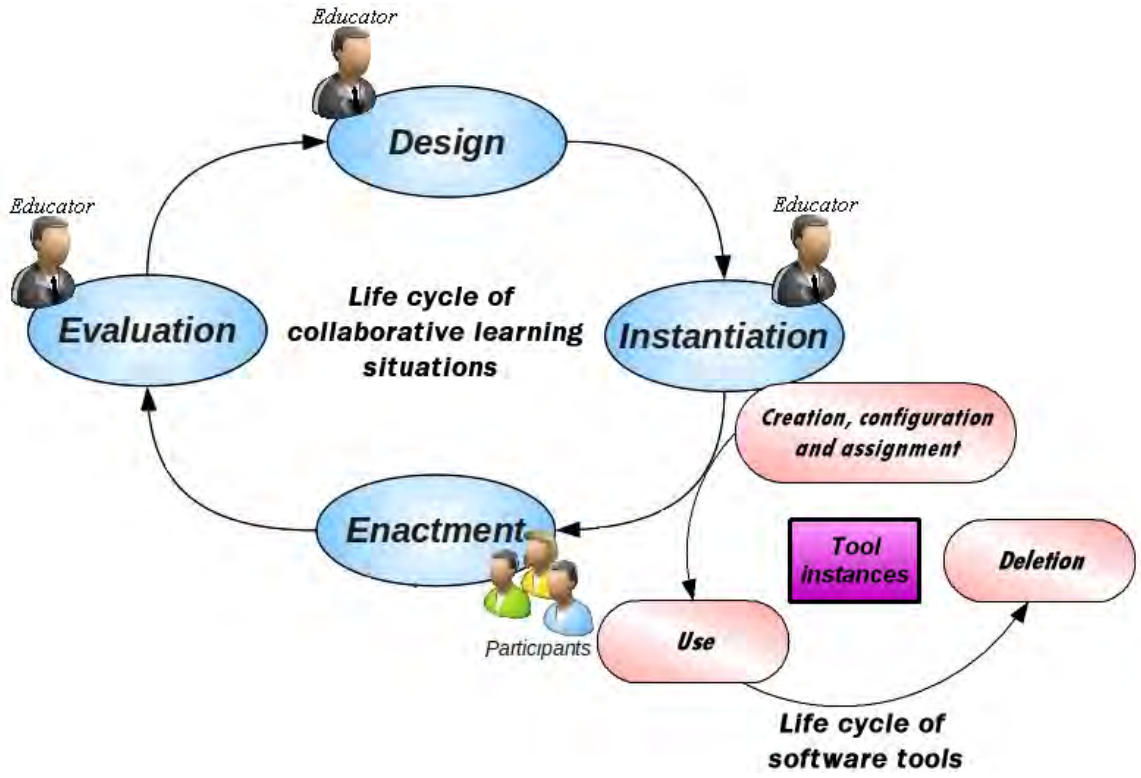

Figure 2.10: Life cycle of collaborative learning situations and life cycle of software tools.

To conclude this discussion, the need for considering the tool life cycle when supporting the instantiation and enactment of collaborative learning situations is remarked. As studied, this life cycle comprises: the creation of tool instances (i.e. creating a new resource and packaging the client side software with a pointer to the resource), their configuration (i.e. changing some of the properties of the resource), their assignment (i.e. distributing the instances among participants, according to their social configuration), their use (i.e. downloading and executing the client side software and accessing the resource it points to), and deletion (i.e. destroying the underlying resource and removing the client side software package, if needed). It is noteworthy that VLEs natively support this life cycle for their built-in tools. Therefore, those approaches extending the set of built-in tools or integrating external tools should take this tool life cycle into account.

2.5. The integration problem

As discussed along this chapter, the restricted set of VLE built-in tools is an important limitation for the support of collaborative learning situations [Bow11]. Practitioners, however, are employing thousands of software tools outside of VLEs for the support of individual and collaborative activities, mainly due to the large increase in the number of SaaS and Web 2.0 applications in the last few years. The integration of external tools is a recent research line aimed at overcoming the limitation of VLEs regarding built-in tools. Those works tackling the integration problem should consider the life cycle of collaborative learning situations and the life cycle of software tools. Unfortunately, despite the number of important works tackling this problem, none of them have proposed a general and widely adopted solution to this problem, as later discussed in section 2.6.

Though the integration of external tools is the path followed in this thesis for tailoring [Mor00] the set of VLE built-in tools, other tailoring alternatives have been followed in the literature. The term tailorability formally refers to the opportunity to adapt generic software applications to the specific end-user's needs and practices in the usage context, rather than in the development context [Mor00]. Therefore, tailorable applications are those that provide mechanisms to modify their appearance and functionality according to end-users' needs. Morch [Mor95] defines three levels of tailorability:

- *Customization.* This is the simplest level of tailorability in software applications. The customization allows end-users to modify the appearance of a system, and to configure the existing functionality. However, the customization does not support the addition of new functionality, nor the development of new code. An example of customization occurs when educators define the questions and select the number of attempts allowed in the Moodle built-in quiz tool.

- *Integration.* The integration allows the addition of new functionality through the connection of existing software components. These componentes are modular pieces of code, whose implementation may range from low level commands to high level applications. The integration typically requires some extra code to achieve interoperability between components, although it must not modify the original code of these components. This code is never developed by end-users who, however, must add it to their systems (e.g. downloading and installing it). An example of this level of tailorability is the integration of the Facebook social networking site in Moodle through the development of a specific Moodle module[67]. End-users (or system administrators) must add this module to their Moodle

[67] `http://moodle.org/mod/data/view.php?d=13&rid=3316`. Last visited: June 2012.

installations, so that it becomes available when designing, instantiating and enacting new activities within this VLE.

- *Extension.* The extension adds new functionality to an application through the modification of its implementation. Therefore, this kind of tailorability allows the realization of major changes in a system, modifying the existing code or adding some new. Besides, in this case, end-users are responsible for the development of this code; this is a very strict restriction that can be met in very few application domains. Examples of this level of tailorability are those tools developed by the own Moodle users from scratch, using its extension interface. The Doodle-like tool for Moodle[68], which replicates the Doodle poll functionality, is an example of extension for this VLE.

In the context of this dissertation, the customization is discarded as a potential solution, since it does not allow the addition of new tools to VLEs. The extension is also excluded, insofar as it requires the development of new tools from scratch by end-users; most VLE end-users are educational practitioners, such as educators and learners, without a technical background, and so it should not be expected them to develop new tools, as a general rule. This dissertation, however, aims at adding new tools to VLEs through the connection of existing components, but *without requiring end-users to develop code*, thus following the integration level of tailorability. It is noteworthy that the integration of existing external tools may, however, *require the programming of some code by developers* in order to provide functional and technological interoperability between VLEs and tools, due to the existing heterogeneity of integration contracts (see section 2.5.1). According to the definition of integration, *this code must not modify the original VLE or tool implementations*, and so, it must be built following the so-called extension interfaces, which enable both the extension of the functionality and the integration with other systems. To avoid misunderstandings due to this name, it is convenient to remark that the purpose of this work is the integration, and not the extension, as previously discussed.

The remainder of this section deeps in the theoretical aspects that hinder the integration of external tools in VLEs, highlighting the role of integration contracts, as an outstanding concept in this problem. After that, the requirements of the main stakeholders interested in the integration of external tools in VLEs are formally defined; those designing and developing integration approaches are advised to take into account these requirements. Then, two kinds of integration approaches that may be useful to tackle this problem are introduced: the standardization of some integration contracts to be adopted by VLE and tool providers, and the proposal of software architectures aimed at adapting the differences among current heterogeneous contracts. Finally, a discussion about the main issues and alternatives that should be considered when designing and implementing any of these approaches is carried out.

[68] `http://moodle.org/mod/data/view.php?d=13&rid=4528`, Last visited: June 2012.

2.5.1. Integration contracts

The overall problem of integrating multiple external tools in multiple VLEs can be abstracted as depicted in Figure 2.11: on the left side, a set of m VLEs with different APIs, architectures and features that *practitioners* (educators and students) may use; on the right side, a set of n tools with different technologies, interfaces, and developed for a varied range of purposes, that *practitioners* may be interested in. Each *VLE provider* may impose different requirements to integrate a tool. Each *tool provider* may also impose different requirements to be integrated in a VLE. The requirements imposed by *VLE and tool providers* to enable the functional extension and the technological interoperability of VLEs and tools represent what has been termed as the **integration contract** [Ghi06], being it explicit or not. An integration contract determines, at least, the technologies, the interfaces and the data models that must be employed to enable the communication between a system and the software application intended for integration with that system [Ala10a]; this contract may also determine the functionality and features supported by the integrated systems. An example in the VLE side is the LAMS Tool Contract[69], which specifies the "behaviours, URLs and API calls that a LAMS tool has to implement to talk to the LAMS Core". On the tool side, for instance, the Google Data Protocol[70] is a REST-based contract used to provide external access to data and functionality of many Google tools.

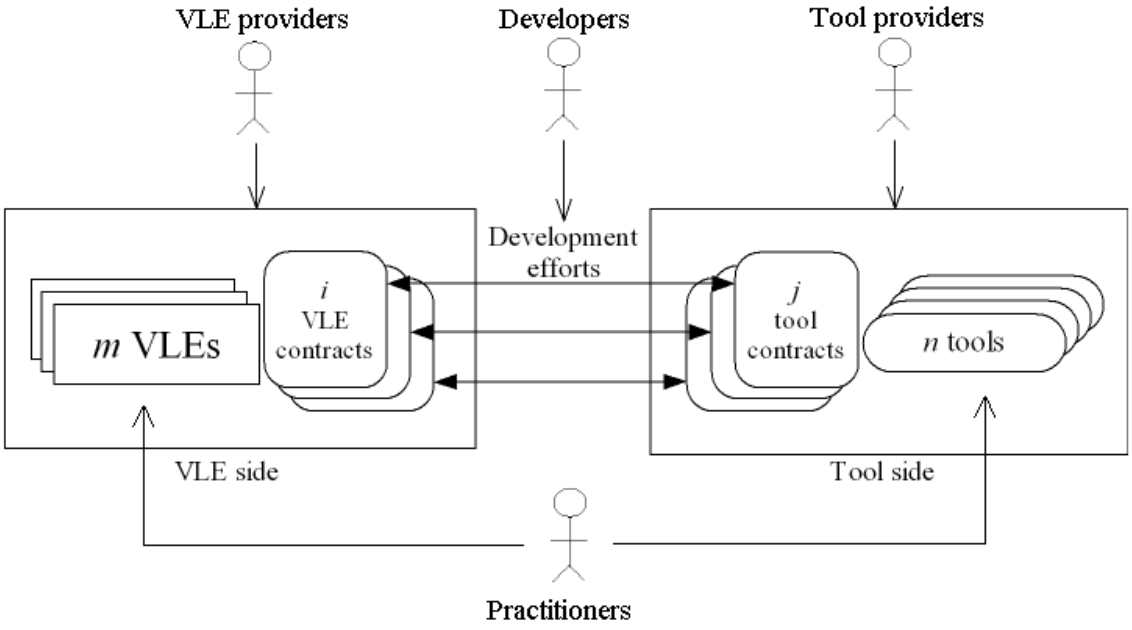

Figure 2.11: Abstraction of the integration problem.

[69] `http://wiki.lamsfoundation.org/display/lams/Tool+Contract`. Last visited: June 2012.
[70] `http://code.google.com/apis/gdata`. Last visited: June 2012.

Each tool may offer one or more integration contracts. These contracts can be specific to one single tool (e.g. Delicious provides a REST-based contract, different from the one offered by Slideshare), or shared by several ones (e.g. the Google Data Protocol). The same can be said for VLE contracts, though these are most frequently VLE-specific. Therefore, i VLE contracts for the m VLEs, and j tool contracts for the n tools should be considered when analyzing the integration problem. The wide variety of technologies, interfaces and data models employed in current VLEs and tools (see sections 2.3 and 2.4) make the proposal of integration approaches, especially those tackling the generic integration of VLEs and tools, a challenging task.

The differences between a VLE contract and a tool contract entail a certain *integration cost*. This cost may include an administrative burden to enable the discovery of the new tool within the VLE, an educational burden to add and configure the tool in a particular learning setting, or a learning burden to get educators and students used to the new tool. Nevertheless, due to the functional and technological heterogeneity between VLEs and tools, most of the integration cost corresponds to the **development effort** required to generate a certain source code. This code allows the functional and technological interoperability between the VLE and the tool, that it is to say, adapts the requirements imposed in a VLE contract to those imposed in a tool contract [Ala10a]. The development of this code cannot be automated because of the existing heterogeneity and particularities of VLEs and tools, and so, a human *developer* must undertake that development effort. The role of *developer* can be played by anyone willing to accomplish the integration; for example the VLE or the tool provider themselves, the practitioners (rarely), or a third-party. Developers normally expect a benefit in return, which could be recognition, reputation, economic compensations or the satisfaction to use or let other use the integrated tools and VLEs [Ala10a].

At this point, it seems reasonable to think that the higher the number of tools integrated in a given VLE, the higher the number of individual and collaborative learning activities that can be performed within this VLE, and so, the higher the number of practitioners that will choose it for their courses. Nevertheless, it is also important to mention that some of these practitioners may be reluctant to change the VLE they are accustomed to, and that some institutions may impose the use of specific VLEs. Therefore, generic integration approaches should be designed with the objective of integrating *the higher number of tools in, at least, the most popular VLEs* [Ala10a]. Nonetheless, since most VLE and tool providers define their own integration contract, then this objective could imply a huge development effort that would not be worthwhile to be made, even by a very populated community of developers. It is noteworthy that the motivation and commitment of developers are key issues to tackle the integration problem, since many previous works, such as Gridcole [Bot08] or CCSI [Vog06] failed in adoption because they could hardly persuade developers to assume the development effort demanded, thus failing in the generation of a critical mass of integrated systems to be used by practitioners through these approaches.

2.5.2. Requirements of the main stakeholders

The main stakeholders interested in the integration of external tools in VLEs are: *educational practitioners*, who actually want to use external tools integrated in VLEs in order to enrich the learning activities that can be realized; *developers*, who write the code needed for the integration of external tools in VLEs; and *VLE and tool providers*, who provide the VLEs and tools that are integrated. The main requirements of these stakeholders have been briefly outlined along this section, although they are formalized next. A summary with the main stakeholders' requirements is shown in Table 2.4.

Table 2.4: Requirements of the main stakeholders.

Stakeholder	Tag	Requirement
Practitioners	**REQ1**	Enable the instantiation of individual and collaborative activities that require the integration of external tools with an attainable effort for educators.
	REQ2	Enable the enactment of collaborative activities that require the integration of external tools, facilitating the collaboration among participants.
	REQ3	Support the integration of existing and popular VLEs and tools.
	REQ4	Support the integration of many external tools.
Developers	**REQ5**	Demand an attainable development effort for the integration of tools and VLEs.
VLE and tool providers	**REQ6**	Be built over existing VLEs and tools, rather than modifying their implementations.

- *Educational practitioners* would generally like to employ integrated tools, as they do with built-in tools. That includes benefitting from the main VLE features, among which, the support of collaboration and groupwork are outstandingly reported [Bow11], as discussed in section 2.3. Therefore, those approaches tackling the integration problem should consider the life cycle of collaborative learning situations (see section 2.2.1), in particular the instantiation and enactment phases, when integrating external tools in VLEs. Integration approaches should thus **enable the instantiation of individual and collaborative activities that require the integration of external tools with an attainable effort for educators**; they should also **enable the enactment of these activities, facilitating the collaboration among participants**. In addition, practitioners would not normally like to give up the VLEs and tools they are used to [Ala10a]. Therefore, integration approaches should **support the integration of existing and popular VLEs and external tools**. Finally, practitioners would like to have the highest number of tools available, in order to support a wide range of learning activities [Bow11]. Integration approaches should thus **support the integration of many external tools**.

- *Developers* are less likely to write the code needed for the integration of tools and VLEs if a high development effort is required. Integration approaches should thus **demand an attainable development effort for the integration of tools and VLEs** in order to encourage the contributions of developers.
- *VLE and tool providers* are rarely willing to modify their systems to comply with an integration approach. Often, they will also disapprove that others modify their systems, since that may cause incompatibilities with the official releases of their VLEs and tools. Thus, integration approaches should **be built over VLEs and tools**, rather than modifying their implementations.

2.5.3. Integration approaches

Two kinds of integration approaches have been typically employed in the literature to tackle the integration problem. Some works have tried to define standard integration contracts, expecting them to be adopted by VLE and tool providers. Others, however, have proposed architectural approaches aimed at adapting the existing contracts through the development of some software elements.

Standardized integration contracts

The definition of one or at least a few standardized contracts could simplify the integration problem. However, nowadays there are not standard contracts that are widely accepted by the main actors in producing and integrating educational software. This can be seen as a consequence of two main factors that normally appear in those fields where conflict of interests and technology are involved [Rob00]: *political disagreements*, and *fast changing technologies*. Historically, each provider decided to make different design decisions when implementing their VLEs and tools. These decisions included, for instance, different technologies, programming languages, or pedagogical methods, thus affecting the way the life cycle of collaborative learning situations was implemented, as discussed in section 2.3.2. As an example, LAMS, which was released only two years after Moodle, was implemented in Java following a scripting activity-based approach, while Moodle had been previously implemented in PHP following a more unsequenced course-based approach. Long time after, when international organisms and consortia tried to produce common standards for the integration of external tools in VLEs, like IMS LTI [IMS06c], VLE and tool providers were reluctant to adopt these standards, mainly due to the effort required to adapt their contracts to these new ones. Nonetheless, the interest on interoperability standards might be changing, since in the last two years several VLE providers like Moodle or Blackboard have adopted Basic LTI [IMS10b], a loosely-coupled contract also proposed by the IMS consortium. However, very few Basic LTI compliant tools are officially reported (about 20 at the

time of writing)[71], being main providers like Google not interested yet. Interestingly, IMS is still working in the IMS LTI standard, and a final and renewed version of IMS LTI (1.1), which also includes Basic LTI, has been released on March 2012 [IMS12]. Nevertheless, the compliance to this new version of IMS LTI is still very low with only six tools and two LMSs listed (according to the same reference that the one in the case of Basic LTI).

Fast changing technologies also hinder the adoption of frameworks or specifications by main vendors, since these frameworks and specification may be replaced in a short time by others that include new technological trends. That happened for example to Gridcole [Bot08], which adopted grid services [Fos98] and the WSRF specification [OAS06] as the basis for the integration of external tools. Nevertheless, soon after Gridcole, the interest in developing tools as grid services passed the hype, while other web technologies became more trendy. Even before, other technologies and standards for software component interoperability, like CORBA (Common Object Request Broker Architecture) [Vin97], DCOM (Distributed Component Object Model) or XML-RPC (Extensible Markup Language - Remote Procedure Call) [StL01], gained momentum, but by the time the problem of integrating external tools in VLEs raised a great interest, they could be considered outdated, since no VLE and very few tools were implementing them.

In conclusion, the combination of these two factors in this context, as well as the high existing heterogeneity in VLE and tool contracts, results in a great challenge behind those trying to standardize new integration contracts to be adopted by VLE and tool providers.

Architectural approaches

The alternative to the proposal of standard contracts to be adopted by VLE and tool providers is the proposal of architectural approaches that adapt the existing VLE and tool contracts without modifying their implementations. Two kinds of architectures have been traditionally employed in the literature to tackle the generic integration of external tools in VLEs. Figure 2.12 depicts these two architectures. These architectures are recurrently found in the related works, which are later studied in section 2.6.

- *m VLE and n tool contracts adapted through adapters* (Figure 2.12a). This kind of architecture allows a direct interoperability between each VLE and each external tool (*one-to-one integration*). This interoperability is achieved by means of the well-known *adapter design pattern* (wrapper) [Gam95], which wraps VLEs and tools, connecting their heterogeneous contracts. Each adapter homogenizes the communication between each VLE and each tool, but without modifying the original source code. Architecturally, two split pieces of code could be added in both the VLE and the tool sides (as it is represented in the Figure 2.12a),

[71] `http://imsglobal.org/cc/statuschart.cfm`. Last visited: June 2012.

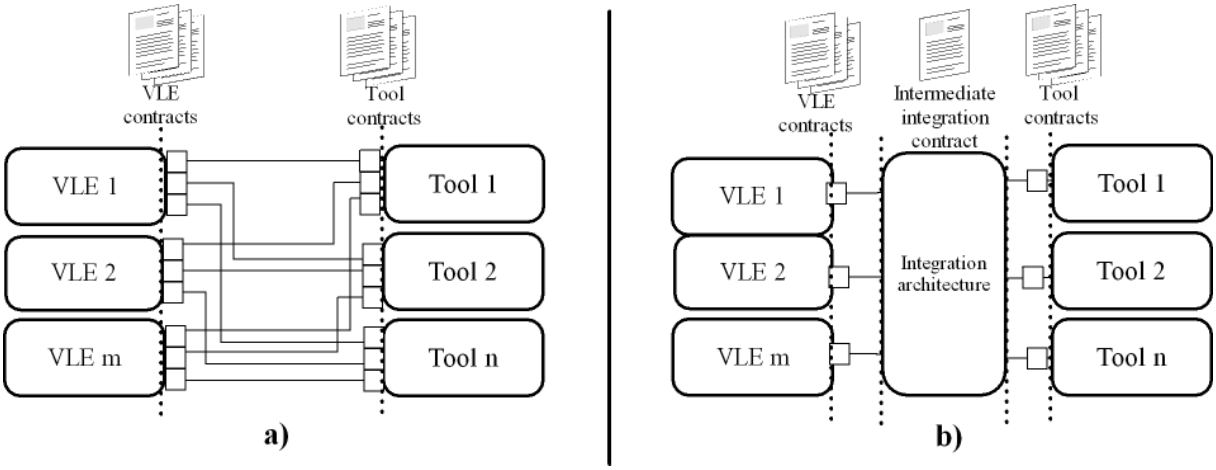

Figure 2.12: Architectural approaches for the integration of tools in VLEs: a) m VLE and n tool contracts adapted through adapters; b) m VLE and n tool contracts adapted through an intermediate software layer and adapters.

or similarly, one single component could package all the code needed for the integration. In both cases, these components could run on third-party domains, unless noted otherwise in VLE or tool contracts. An example of this approach can be found, for instance, in those Moodle modules that integrate external tools, like the Big Blue Button web conferencing module[72]. Nevertheless, due to the heterogeneity in contracts, very little of the developed code or the acquired experience can be reused in order to integrate the same tool in other VLEs, and vice versa.

- *m VLE and n tool contracts adapted through an intermediate software layer and adapters* (Figure 2.12b). This kind of architecture enables the interoperability between multiple external tools and multiple VLEs (*many-to-many integration*) through a common middleware integration element. This common integration element could be made up of different software components, which may run on different domains. Thus, a new intermediate integration contract that enables the communication of VLEs and tools with these components has to be defined. In this context, the adapter pattern could also wrap VLEs and tools in order to meet the requirements imposed by this new intermediate contract (as it is represented in Figure 2.12b). This architecture reduces the development effort needed to integrate each tool in each VLE, since the required code is partially implemented in the common integration element. Nevertheless, the degree of integration and functionality that can be achieved with this architecture is lower compared to the first one, as it is further discussed in the next section. Significantly, many recent integration works in literature, like GSI [Fue11] or Apache Wookie [Wil08] are following this kind of architecture.

[72] `http://moodle.org/mod/data/view.php?d=13&rid=3524`. Last visited: January 2012

2.5.4. Design issues and alternatives

The definition of new approaches tackling the integration of external tools in VLEs is a challenging task, as it was discussed along this section. Therefore, before embarking on this task there are three main questions that those addressing this problem should answer:

- **What/Where to integrate?** What tools to integrate and where (in which VLEs) are they integrated? This question refers to the set of tools and VLEs that are eligible for integration, according to the proposed approach. Due to the existing heterogeneity in integration contracts, some technical and functional *restrictions* on VLEs and tools must always be defined; these restrictions may exclude some VLEs and tools from being integrated.
- **How to integrate?** How tools are integrated in VLEs? The aforementioned kinds of integration approaches foster different integration *multiplicities* (e.g. one-to-one and many-to-many integrations). Besides, each particular integration can be designed pursuing a higher or a lower *degree of software coupling* between the tool and the VLE (i.e. a more or less coordinated and interdependent interoperability between the tool and the VLE).
- **What does the integration allow to do?** This question refers to how much of the tool *functionality* can be controlled within the VLE. This functionality normally comes from the original features offered by VLEs and tools, although the integration software may add extra functionality on top of the one offered by VLEs and tools. Remarkably, different integration solutions may offer access to the same tool from the same VLE, but with supporting a different functionality.

The main technical and functional issues related to each of these three questions and their corresponding alternatives are discussed next. New integration approaches should deal with these issues, while trying to satisfy the stakeholders' requirements. Nevertheless, at the time of making a decision on the alternatives, it should be noted that some of the issues are interrelated, and that different requirements may be conflicting.

What/Where to integrate? - Restrictions on VLEs and tools

One relevant technical issue, which affects the eligibility of VLEs and tools, refers to the *number of restrictions* imposed by integration approaches to VLEs and tools. Some examples of these restrictions may be programming languages (e.g. PHP for Moodle), certain frameworks (e.g. OpenACS for .LRN), or exchange data models (e.g. RSS-based models in most blog tools), among others. Imposing many restrictions usually allow richer, particularized interactions between VLEs and tools, although this alternative may exclude interesting VLEs and tools from

being integrated. For instance, IMS LTI [IMS06c] imposes many restrictions for the sake of a richer communication between VLEs and tools. On the contrary, approaches including few restrictions generally reduce the interactions supported in the communication of VLEs and tools, but also reduce the development effort required to meet these restrictions; as a consequence, the chances that some popular VLEs and tools may be discarded for the integration with a certain approach (due to an unattainable development effort) are reduced too. In particular, as IMS LTI requires a significant development effort and has not been widely adopted, a new contract termed IMS Basic LTI [IMS10b] was proposed with less restrictions aiming at reducing this effort.

A related technical issue is the *degree of adoption of the restrictions*. The more widespread the restrictions imposed by an integration approach, the more the number of VLEs and tools that are likely to natively meet them. Those VLEs and tools that natively meet these restrictions are normally integrated with a lower effort, and without modifying their original implementations. Nowadays, for instance, the popularity of the REST style [Fie00] is quite spread, offering Google and other major tool providers REST-based contracts. In contrast, the less widespread the restrictions imposed by an approach, the fewer the VLEs and tools that are likely to be integrated through it. For example, Gridcole [Bot08] can only integrate tools developed as grid services following the WSRF [OAS06] specification, which never got too many adopters; as a consequence, the number of existing tools that can be integrated in Gridcole is very limited. Interestingly, *ad hoc* integration approaches might want to impose restrictions with a low adoption, since their objective is the integration of a specific tool in a specific VLE (for which these restrictions might be meet, despite not being very popular).

How to integrate? - Multiplicity and software coupling

Another important technical issue is the *multiplicity of the integration*. Generic integration approaches are typically designed to promote the integration of multiple tools in multiple VLEs (many-to-many integration). These approaches may reduce the development effort by fostering a high code reuse among integrations, but at the cost of supporting only generic functional commonalities, due to the existing heterogeneity in VLE and tools contracts. For example, the GSI architecture [Fue11] was designed to foster a many-to-many integration with IMS LD compliant platforms and third-party services. *Ad hoc* integration approaches, however, typically foster a one-to-one integration, and so, they can achieve a richer communication between VLEs and tools. Nevertheless, an *ad hoc* approach is normally useful only for the integration of one tool in one VLE. Many examples of *ad hoc* approaches can be found in existing Moodle modules or Blackboard Building Blocks. Alternatively, other approaches could be designed to facilitate the integration of multiple tools in the same VLE (one-to-many integration), and less often, the integration of the same tool in multiple VLEs (many-to-one integration).

The *degree of software coupling* [Dha95] is another technical issue that is closely related to the restrictions and multiplicity issues. A tight integration (high software coupling) allows to control the behavior of the tool from the VLE to a higher extent, but in exchange for a significant additional development effort. This additional effort is due to the interdependency between software components. Many *ad hoc* integration approaches, like Sloodle [Liv08], which integrates the virtual world Second Life[73] in Moodle, have chosen a tight approach, aiming at a higher coordination between the tool and the VLE. On the other hand, a loose integration (low integration coupling) generally entails a lower control on integrated tools, but also reduces the code needed to allow the communication between each VLE and each tool. For example, those supporting REST services consider this a good example of a loosely-coupled technology [Vin07], and so, works based on REST services, like Apache Wookie [Wil08], can leverage the advantages of this loosely-coupling.

What does the integration allow to do? - Functionality offered

The *degree of functionality offered* by an integration approach (how much of the functionality of the tool can be controlled from the VLE) is an important functional issue that is related to the aforementioned technical issues. Offering a high degree of functionality normally requires additional restrictions, a higher coupling, and possibly promoting a one-to-one integration. For instance, Moodlerooms[74] has developed an integration that allows Moodle users to access Google Apps for Education[75] within Moodle, enabling a higher control on Google Apps; however, this approach requires a large amount of code that is useful only for this VLE. On the other hand, approaches offering a low degree of functionality generally demand a lower development effort, and possibly imposing less restrictions. IMS Basic LTI [IMS10b] offers the lowest degree of functionality in existing approaches, providing just a single common representation of each integrated tool.

Besides, in this context, the *management of the life cycle of software tools* (see section 2.4.2) can be considered an important functional issue, which stems from the degree of functionality offered, since most VLEs and many tools are designed to support individual and collaborative activities, as discussed along this chapter. This issue refers to the creation, configuration and assignment of different instances for each group in each learning activity, which can be particularly cumbersome in complex collaborative learning flows that involve multiple groups and tools. Integration approaches, such as Gridcole [Bot08], GSI [Fue11] or Apache Wookie [Wil08], aimed at supporting the management of the tool life cycle, facilitating practitioners the instantiation and enactment of collaborative learning situation. On the contrary, approaches like IMS

[73] `http://secondlife.com`. Last visited: June 2012.
[74] `http://moodlerooms.com`. Last visited: June 2012.
[75] `http://google.com/apps/intl/en/edu`. Last visited: June 2012.

LTI [IMS06c] or Basic LTI [IMS10b] that do not consider this issue hinder and even preclude from instantiating and enacting many individual and collaborative activities, even though these activities may be of great interest for practitioners.

2.6. Analysis of existing integration approaches

There are many proposals in the literature tackling the integration of external tools in VLEs. This section discusses the most relevant works and relates them with the design issues and alternatives that have been introduced in the previous section. A summary with the issues, alternatives and examples is shown in Table 2.5.

Table 2.5: Design issues and alternatives chosen for the integration of external tools in VLEs.

	Technical issues				*Functional issues*	
	Number of restrictions imposed	*Degree of adoption of the restrictions*	*Multiplicity of the integration*	*Degree of software coupling*	*Degree of functionality offered*	*Management of the tool life cycle*
Moodle, LAMS, Sakai or Blackboard (VLE) contracts	many	low	one-to-many	*	*	*
Gridcole [Bot08], Pelican [Vel09] (VLE) contracts	some	low	one-to-many	medium	medium	yes
Moodle Web Services SOA [Con09]	some	medium	one-to-many	medium	medium	no
Moodlerooms	some	medium	one-to-many	medium	high	yes
Sloodle [Liv08]	many	low	one-to-one	tight	high	yes
Google Apps, Delicious or Zoho (tool) contracts	some	high	many-to-one	loose	*	*
IMS LTI [IMS06c], OKI [Col02]	many	low	many-to-many	tight	high	no
CCSI [Vog06]	many	low	many-to-many	tight	high	yes
PoEML-based architecture [Fon09]	some	low	many-to-many	medium	low	yes
GSI Architecture [Fue11]	some	medium	many-to-many	medium	low	yes
Apache Wookie [Wil08]	few	medium	many-to-many	loose	low	yes
Basic LTI [IMS10b]	few	high	many-to-many	loose	very low	no

*The particular approach employed to integrate these VLEs or tools may condition the alternative chosen for this issue.

The most widely adopted VLEs, Moodle, LAMS, Sakai or Blackboard impose technologically heterogeneous and specific VLE contracts with many restrictions on the tools that can be

integrated. Therefore, in most cases, a considerable development effort is required to integrate the same external tool in several of these VLEs, due to the use of the first architectural approach (one-to-one) presented in section 2.5.3. The exact amount of effort is highly dependent on the degree of coupling supported and on the degree of functionality offered, as it is further studied with illustrative examples in the evaluation chapter of this document. Interestingly, due to the high number of users and active sites of VLEs like Moodle, many developers have considered that it is worthwhile to develop an *ad hoc* piece of software to integrate one tool in this VLE. For instance, members of the European EduJudge project[76] developed a module to integrate QUESTOURnament in Moodle [Reg09]. In line with this, some remarkable tools that were integrated via Moodle Modules are the Tiddlywiki personal notebook, the Facebook social networking site or the Kaltura[77] video platform. However, the existence of these modules cannot be generalized, and other VLEs with a lower adoption like LAMS or .LRN have found hardly any external developer willing to accomplish the integration of new tools in these platforms. Actually, one of the few examples of tools integrated in these other VLEs that can be found in the literature is <e-Adventure> [Bla10], an authoring platform for the design of educational games that was first integrated in Moodle, and afterwards in LAMS [Bla10].

Other VLEs, such as Gridcole, or Pelican also impose very specific VLE contracts, but requiring less restrictions and promoting a lower degree of integration with external tools (compared to the main VLE providers). Nevertheless, these restrictions, which are the use of grid services and SOAP-based services [Vin02], do not reflect the current trends regarding distribution technologies. Therefore, these restrictions along with the idea that Gridcole and Pelican were conceived to replace other commonly used VLEs, hindered their embracement by educators and students, and also the contributions from external developers.

Some works, like the one by Conde-González [Con09], decided to extend the contracts of existing VLEs in order to facilitate the integration of external tools. In this particular example, the authors extended the Moodle integration contract through a Service-Oriented Architecture (SOA) [Vin02], whose objective was to promote the integration of external web services. This approach simplifies the integration process, since the development effort has been partially undertaken by the authors in the VLE side. However, this solution cannot be easily extrapolated to other VLEs. Besides, not many tools are currently complying with the restrictions imposed by this approach, thus hindering their adoption.

Some other authors decided to implement their own *ad hoc* solutions aimed at offering a high degree of functionality just for certain specific tools and VLEs. This is the case of Sloodle, whose authors proposed an *ad hoc* tight integration architecture [Liv08], or the aforementioned commercial product for the integration of Google Apps in Moodle, Moodlerooms, which defined

[76] `http://eduvalab.uva.es/en/projects/edujudge-project`. Last visited: June 2012.
[77] `http://kaltura.com`. Last visited: June 2012.

a proprietary approach, seeing that its providers do not rely on external contribution. The problem, once again, is that these solutions cannot be easily generalized and applied to other VLEs and tools.

On the tool side, it is noteworthy that most software tools have not been designed to be integrated in other systems. Thus, they implicitly offer specific tool contracts, whose restrictions did not consider current trends on existing VLEs. In this context, many SaaS [Tur03] tool providers have started to offer generic loosely-coupled contracts through web technologies and REST interfaces in the last few years. This is the case of Delicious, Google Apps for Education, or Zoho[78], among many others. Generic loosely-coupled tool contracts normally avoid low level details regarding the features offered by tool interfaces, thus limiting the set of interactions and configurations of these tools that can be exploited. Therefore, the functionality available when integrated these tools in VLEs is normally low, although this functionality highly depends on the specific integration approach.

The first remarkable works that tried to standardized some contracts as a general many-to-many solution, in order to promote the integration of existing external tools in existing VLEs, rather than defining new VLEs or tools, or developing *ad hoc* integrations, were the Open Knowledge Initiative (OKI) [Bav03, Col02], IMS LTI and CCSI. These works defined tightly-coupled contracts that offered a high degree of functionality, even though only the latter supports the management of the tool life cycle, through a set of run-time services. The high development effort required by these approaches and the evolution in the technological trends may have deprived OKI, Full LTI and CCSI from a more successful adoption.

Recent generic integration approaches, however, aimed at reducing the number of requirements and the degree of coupling, but at the cost of reducing the functionality offered too. Nevertheless, they normally find some problems to integrate popular VLEs or tools, due to the kind of restrictions they impose. For example, the PoEML-based (Perspective-oriented Educational Modeling Language) architecture [Fon09] is based on a language defined by the authors, named PoEML, [Cae06b], which has not been adopted in practice. Another example is the GSI architecture [Fue08, Fue11], which was designed to extend VLEs that support IMS LD through a medium-coupled intermediate integration contract. Despite the low degree of functionality offered by GSI, the support to the instantiation and enactment of collaborative activities is significantly reported. Nevertheless, only an extension of .LRN can currently comply with the restrictions of GSI, and very few tools have been so far integrated.

Finally, two works have been recently proposed with the aim of promoting the integration of multiple external tools in multiple VLEs, through loosely-coupled approaches. One of these works is the Apache Wookie architecture [Wil08]. This architecture offers a low degree of functionality,

[78] `http://zoho.com`. Last visited: June 2012.

but considering the synchronous communication and the collaboration as outstanding features, and supporting the life cycle of tools integrated through this approach. Nevertheless, Apache Wookie only enables the integration of small applications developed as W3C widgets. This is a very strict requirement that hinders the integration of many other existing tools that may be of interest for practitioners. Actually, none of the Top 100 Tools for Learning meets the W3C Widgets specification [W3C11].

The other recent work is IMS Basic LTI [IMS10b], which proposes a loosely-coupled intermediate contract with a very low degree of functionality, just providing a single standard mechanism for launching external applications for all the participants in the same learning activity, thus failing to alleviate practitioners of the burden of creating, configuring and assigning external tool instances to support collaborative activities. Therefore, unlike Apache Wookie, Basic LTI does not support the life cycle of external tools, thus hindering and precluding from instantiating and enacting many collaborative learning situations.

Before concluding this section, it is convenient to analyze the current trends on the integration of external applications in other platforms that may also be employed in educational contexts, such as wikis, social networking sites, or PLEs, which gained momentum in the last five years. For instance, generally adopted platforms like MediaWiki, Facebook, Google+, and most of the PLEs developed so far enable a loosely-coupled third-party integration through their own specific contracts, similarly to what happens with VLEs. However, most integrated applications are mash-ups, or offer a very lightweight functionality [Sev08]. In this context, Google along with MySpace defined, in 2007, a common specification for the integration of third-party applications in social networks, called OpenSocial [Gre09, Has09], which was developed as an open source reference implementation named Apache Shindig[79]. Despite its success on adoption by about twenty social networking sites [Has11], like MySpace, LinkedIn, Hi5[80], Ning[81], or XING[82], and many other applications, neither Facebook, nor surprisingly Google+, have adopted OpenSocial yet. Unfortunately, this specification cannot be used to tackle the integration of external tools in VLEs, since it does not consider pedagogical approaches, nor the main educational concepts included in VLEs, such as roles, social configurations, structure of activities or courses, among others. In any case, it is noteworthy that OpenSocial, the successful MediaWiki, Facebook, Google+, and most PLEs promote a low degree of coupling with third-party integrated applications. This should be seen as an important trend that can be extended to the cases of VLEs and tools, thus discarding tighter solutions when proposing new integration approaches.

All in all, it can be seen that the integration approaches analyzed here present limitations concerning the kinds of VLEs and tools that can be integrated, the development effort required

[79] `http://shindig.apache.org`. Last visited: June 2012.
[80] `http://hi5.com`. Last visited: June 2012.
[81] `http://ning.com`. Last visited: June 2012.
[82] `http://xing.com`. Last visited: June 2012.

to accomplish the integration, and the support of the tool life cycle. Thus, there is an opportunity for new generic approaches to overcome these limitations. Considering the stakeholders' requirements presented in section 2.5.2, the alternatives recommended for these new approaches would be: to *impose few and popular restrictions* to VLEs and tools; to foster a *many-to-many integration* and a *loose integration* to reduce the development effort; and, to facilitate at least, *the management of the life cycle of external tools*, as part of the compromise that needs to be achieved among the degree of functionality offered and the remaining design issues.

2.7. Conclusions

Collaborative learning is a process in which knowledge is constructed though interactions with other partners, and in most cases, results in a more effective learning compared to individual or competitive pedagogical processes. The use of technology to promote these interactions is researched by practitioners, psychologists, and technologists in the multidisciplinary CSCL research field. VLEs have outstandingly been employed for the support of education and collaboration at all levels during the last few years, since they include significant pedagogical and collaborative features, such as the management of groups, roles, activities and courses. Practitioners are recurrently using VLEs for the design, instantiation, enactment and evaluation of collaborative learning situations within a centralized platform. Nevertheless, the restricted set of VLE built-in tools hinders and even precludes from realizing many individual and collaborative learning activities. Here, the increasing popularity of web technologies and services has caused a growing research interest in the integration of external tools in VLEs, so that a wider range of activities may be supported. This research line replaces a former trend according to which the number of VLE built-in tools was extended by means of the development of new tools from scratch, which were specifically designed and implemented for one single VLE.

Nevertheless, the integration of external tools in VLEs is a very complex problem, mainly due to the existence of a great variety of heterogeneous contracts, and none of the current existing works on this matter is offering a generic solutions with a widespread adoption. Figure 2.13 summarizes the main design issues and alternatives found in current integration works, indicating how they affect to the stakeholders' requirements. By way of summary, approaches imposing a high number of restrictions with a lack of adoption exclude many interesting VLEs and tools from being integrated. Moreover, tight and one-to-one integration approaches normally require a high development effort, thus discouraging external developers to contribute to the integration of new tools and VLEs, this way hindering their potential adoption by practitioners. Finally, approaches lacking support to the tool life cycle hinder the instantiation and enactment of collaborative learning situations, which are commonly employed in educational practices as a way for students to achieve a more effective learning.

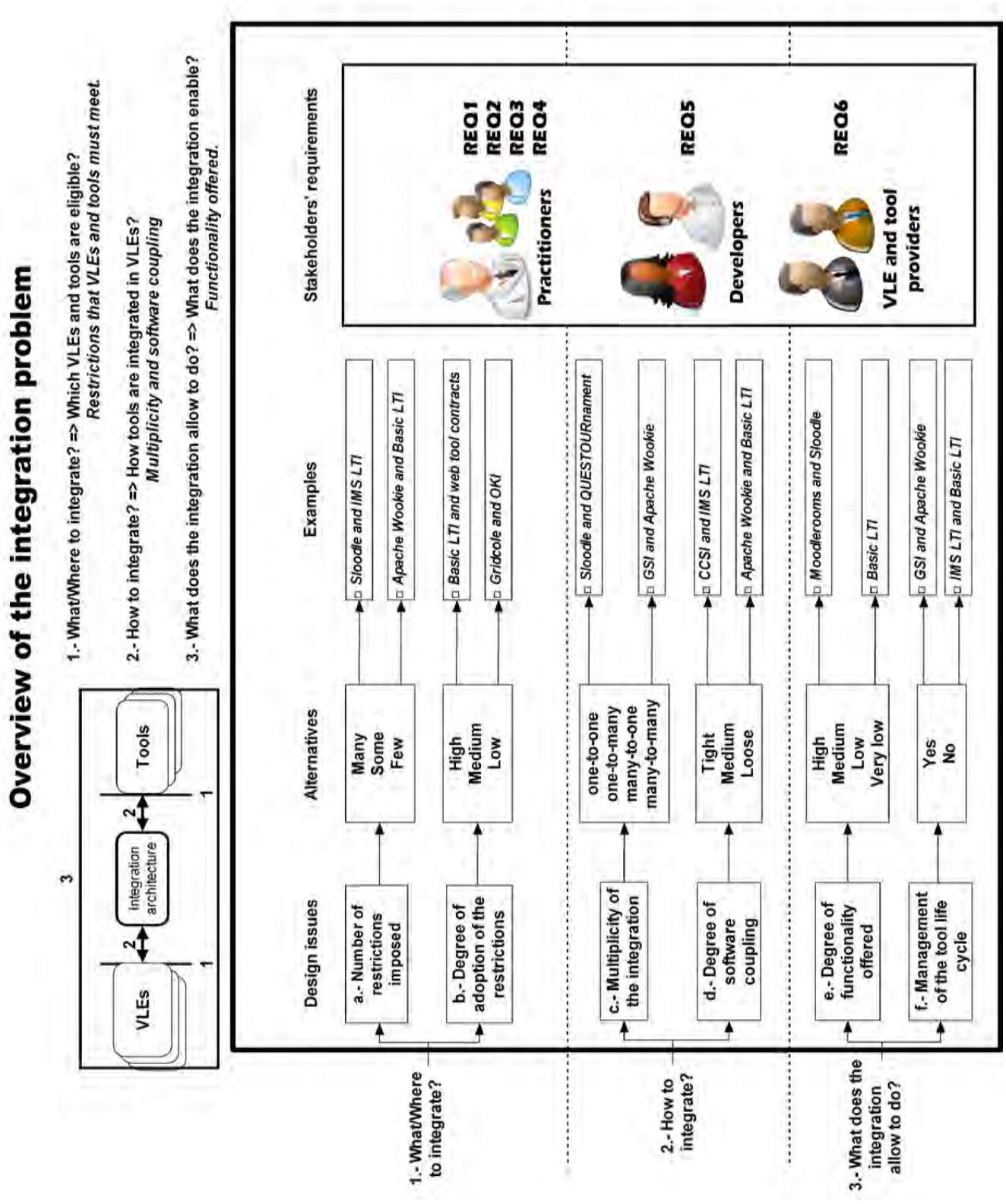

Figure 2.13: Overview of the integration problem: design issues, alternatives, examples, and stakeholders' requirements.

The analysis of the limitations found in existing integration works and their causes, can be distilled as lessons learned of relevance to other researchers that tackle the multiplatform integration of third-party tools, whether in the educational domain or in others. In order to overcome these limitations, the next chapter presents the core contribution of this thesis, an architecture that satisfies the main stakeholders' requirements by featuring: *a low number of broadly adopted restrictions*, thus facilitating the integration of existing and popular VLEs and tools without modifying their code; *a many-to-many and loose integration*, thus promoting the integration of many tools in many VLEs with a lower development effort; and a *sufficient degree of functionality* which, at least, enables the *management of tool life cycle*. The decisions regarding the functional issues are the result of a trade-off considering all the stakeholders' requirements, and take into account that the degree of functionality offered compromises the remaining issues.

Chapter 3

The GLUE! architecture

This chapter tackles the objective of proposing an architecture, called GLUE! (Group Learning Uniform Environment), that enables the integration of multiple existing external tools in multiple existing VLEs, taking into account the main design issues of the integration problem, meeting the main stakeholders' requirements, and overcoming the limitations found in previous related works. Therefore, GLUE! mediates so that VLEs and external tools, which may be developed with different technologies and for different purposes, achieve interoperability. In order to structure the presentation of this architecture in a logical order, section 3.2 first introduces the methodology followed for the proposal of the GLUE! architecture. Then, section 3.3 presents the main requirements considered in the design of GLUE!. These requirements stem from the limitations found in previous integration works, and take into account the issues and alternatives analyzed in chapter 2. After that, section 3.4 describes the proposed architecture, including technical and functional details. Section 3.5 explains the overall behavior of the architecture, detailing the use cases that are currently supported. Then, security issues are discussed in section 3.6, explaining the problems found in this particular context, and offering a compromise solution for these issues. Next, advantages and limitations of GLUE! are discussed in section 3.7. This section also discusses the compliance to the stakeholders' requirements, the compatibility of the architecture with other loosely-coupled approaches in this domain, and the application of its design principles in order to achieve interoperability between software systems in other contexts. Finally, section 3.8 concludes the chapter offering a general picture of the integration problem and the proposed solution, including the requirements and design decisions.

The GLUE! architecture is the major contribution of this dissertation. This architecture, including the initial requirements and the design decisions has been published in [Ala12a]. Besides, the context in which the security issues occur was first studied in [Ala10b], while the final solution presented here is intended to be published in the near future.

3.1. Introduction

The integration of external tools in VLEs is a research trend aimed at increasing the diversity of learning activities that can be instantiated and enacted within VLEs. Nevertheless,

this is a very complex problem, because of the existing functional and technological heterogeneity in VLEs and tools, as discussed in chapter 2. Therefore, the proposal of solutions that tackle the generic integration of external tools in VLEs is a great challenge, and none of the existing generic works in this context has achieved widespread adoption so far.

On one side, the lack of adoption of standards tackling the specific integration of external tools in VLEs was discussed in the previous chapter, being exemplified through IMS LTI [IMS06c], and to a lesser extent (due to a certain success regarding VLE providers, although still quite limited regarding tool providers), through Basic LTI [IMS10b]. On the other side, different architectural approaches found in the literature within the scope of the integration problem, like GSI [Fue11] or Apache Wookie [Wil08], were also presented in the previous chapter, although they have not become very popular either. Three main limitations, namely the restrictions imposed on VLE and tool providers, the high development effort required, and the lack of support to the life cycle of external tools, which is especially relevant in the instantiation and enactment of individual and collaborative activities, were pointed out as the main causes for the lack of adoption of these generic integration works.

A new integration architecture, called GLUE!, which takes into account these limitations, the main stakeholders' requirements, as well as the design issues and alternatives discussed in the previous chapter, is proposed in this chapter. The GLUE! architecture presented here is designed to tackle the loosely-coupled generic integration of existing external tools in existing VLEs, imposing few restrictions with a widespread adoption on VLE and tool providers. That reduces the development effort and facilitates the integration of many well-known tools in popular VLEs. As a consequence the degree of functionality offered has to be low (as a trade-off between the technical and functional design issues analyzed in the previous chapter), although the management of the tool life cycle is supported in order to facilitate the instantiation and enactment of collaborative learning situations, as required by practitioners.

Nevertheless, before proposing a new architecture, it is convenient to study the formal definition of software architectures in computer science [Sha96b]. According to three well-known methodologists on software engineering, Booch, Rumbaugh and Jacobson, a software architecture is defined as "a set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaboration among these elements" [Boo99]. The interfaces of these elements and their expected behavior are formally characterized within operation contracts [Lar02] (being they explicit or not). In the particular context of this dissertation, these operation contracts are normally referred as integration contracts [Ghi06]. Therefore, and according to this definition, the design of GLUE! must determine the structural elements that compose the architecture, their integration contracts, and also their behavior, so that the collaboration among these elements can be achieved.

3.2. Methodology and process

This dissertation follows a global research methodology [Gla95] that combines the engineering methodology [Adr93] and the empirical method (in the evaluation stage) [Big82], as described in the introductory chapter. This global research methodology includes four phases (informational, propositional, analytical and evaluative) that are iteratively covered. The proposal of the GLUE! architecture is the result of this research, and it can be framed within the propositional and analytical phases of this global research methodology. Nevertheless, GLUE! is a technical proposal that can be seen as a software engineering project, and so, a more specific process for the design and implementation of software engineering projects should be followed.

The GLUE! architecture is designed and developed following the **Unified Process (UP)** [Lar02], which is one of the most popular and best documented software development processes [Jac99, Kru04]. The UP was chosen because of its three defining features. First, the UP is an iterative an incremental process, and so, it fits within the global research methodology, which is also iteratively applied. Second, the UP is architecture centric, being software architectures the core over which projects are built. An architecture is exactly the kind of solution proposed in this dissertation to tackle the integration problem. Third, the UP is a use case driven process. Use cases define a system behavior from end-users' perspective [Lar02]. The support of the tool life cycle, which was studied in the previous chapter, represents the behavior practitioners may expect when adding external tools to their collaborative learning situations.

UP projects are organized following four iterative phases: the *inception phase*, where the scope of the project is studied and an investigation is performed to decide whether to go on with the project or not; the *elaboration phase*, where the architecture is iteratively designed and implemented, and high-risk issues are mitigated; the *construction phase*, where low-risk issues are tackled and the environment in which the system must be deployed is prepared; and the *transition phase*, which is focused on testing and deployment. Besides, several traditional disciplines (also referred as activities) are defined in the UP, being they iteratively covered along the four phases. These disciplines include: *business modeling*, *requirements*, *design*, *implementation*, *test*, and *deployment* [Lar02]. For instance, the elaboration phase strongly focuses on business modeling, requirements and design, while the transition phase emphasizes testing and deployment. Nevertheless, it is noteworthy that most disciplines are partially covered in all the phases.

Two important considerations must be done for the application of these disciplines to the proposal of this dissertation. The *business modeling* discipline, which normally comprises a deep understanding of the concepts and scope of the application context, can be included as part of the informational phase in the global research methodology. Actually, most relevant concepts, as well as the context of this proposal were already presented in chapter 2. The

requirements discipline includes the definition of the main stakeholders' requirements, which were also presented in the previous chapter. These are a rather stable set of requirements, since they are the contribution of a complete research analysis, and so, they lead the proposal of the GLUE! architecture from the very beginning. Other classical functional and non-funcional requirements in software engineering projects [Lar02], such as reliability or robustness, are also considered when designing and implementing the architecture, although they are not the main scope of this research work. Interestingly, the global research methodology includes an evaluative phase with multiple experiments involving real practitioners, what might cause the appearance of new functional and non-functional requirements. As a remarkable example, after the evaluation of a first implementation of the GLUE! architecture, a new use case that enables the update of users sharing external tool instances was added in the tool life cycle, and as part of the GLUE! behavior. This new use case aimed at supporting a higher flexibility, allowing educators to adapt their instantiated activities to typical student absences.

Though the UP is an incremental and iterative process, this dissertation only presents the final results, and so, to facilitate the reading, the narration does not follow that incremental and iterative approach. Instead, the initial requirements and the design of the GLUE! architecture are fully explained in this chapter, while the implementation, testing and deployment are presented in the next chapter.

3.3. Initial requirements and design decisions

The initial requirements for the design of the GLUE! architecture are the main stakeholders' requirements identified in chapter 2, according to which:

- GLUE! should **enable the instantiation of individual and collaborative activities** that require the integration of external tools with an attainable effort for educators (REQ1).
- GLUE! should **enable the enactment of collaborative activities** that require the integration of external tools, facilitating the collaboration among participants (REQ2).
- GLUE! should **support the integration of existing and popular VLEs and tools** (REQ3).
- GLUE! should **support the integration of many external tools** (REQ4).
- GLUE! should **demand an attainable development effort** for the integration of new tools and VLEs (REQ5).
- GLUE! should **be built over existing VLEs and tools**, rather than modifying their implementations (REQ6).

The decisions made in the design of the GLUE! architecture are intended to meet these requirements. Besides, these decisions also take into account the lessons distilled in chapter 2, after analyzing the integration problem and the related works. Table 3.1 shows an overview of the design issues, and the alternatives taken in the case of GLUE!. The choice of these alternatives is further explained next:

Table 3.1: Alternatives to the design issues chosen for the GLUE! architecture.

Questions	Design issues	**Alternatives chosen**
What/Where to integrate?	a) Number of restrictions imposed	**To impose a low number of restrictions**
	b) Degree of adoption of the restrictions	**To impose popular restrictions**
How to integrate?	c) Multiplicity of the integration	**To foster a many-to-many integration**
	d) Degree of software coupling	**To promote a loose integration**
What does the integration allow to do?	e) Degree of functionality offered	**To support the management of the tool life cycle**
	f) Management of the tool life cycle	

a) **To impose a low number of restrictions**. Practitioners worldwide are currently using tens of VLEs and thousands of software tools. The lower the number of restrictions imposed, the more VLEs and tools that could be eventually integrated. Some previous works did not succeed because they imposed a high number of restrictions, thus precluding many VLEs and tools from being integrated. Therefore, it seems advisable to design the GLUE! architecture to impose a low number of restrictions on VLE and tool providers.

b) **To impose popular restrictions**. Imposing popular restrictions facilitates the integration of many tools and VLEs that may meet them natively. Therefore, to overcome the limitation of some previous works that impose restrictions with a low degree of adoption, the GLUE! architecture must be designed to impose only popular restrictions on VLE and tool providers.

c) **To foster a many-to-many integration**. The multiplicity of an integration approach has an impact on the development effort required to integrate multiple tools in multiple VLEs, and thus, it conditions the number of external developers that may want to collaborate to increase the set of integrated tools and VLEs. Therefore, to avoid requiring a high development effort when integrating a set of tools in different VLEs, as it happens in some of the previous one-to-one integration works, the GLUE! architecture must be designed to foster a many-to-many integration.

d) **To promote a loose integration**. The degree of software coupling has also an effect on the development effort required to integrate each tool in each VLE. Actually, some previous

works demand a high effort because of their tight integration approaches. With the aim of demanding a lower development effort, the GLUE! architecture must be designed to promote a loose integration of external tools in VLEs.

e) **To support the management of the tool life cycle**. In the context of this dissertation, main VLEs and tools have been designed to support collaboration and groupwork, as discussed in the previous chapter. Nonetheless, there are some integration approaches that do not consider the life cycle of collaborative learning situations, nor the tool life cycle in collaborative settings. In consequence, practitioners must assume a significant burden to instantiate and enact their learning activities. In order to overcome this limitation, the GLUE! architecture must be designed to support the management of the tool life cycle in the instantiation and enactment of individual and collaborative activities.

These five design decisions are oriented to reduce the adoption barrier of the architecture in two different ways. Regarding educational aspects, they increase the number of VLEs and tools that can potentially be used by educators and students. Besides, they preserve the main collaborative features of VLE built-in tools, so that practitioners do not need to change their learning practices. Regarding technological aspects, they facilitate the integration of new tools and VLEs, thus encouraging external developers to contribute to the architecture.

3.4. Description of the architecture

This section describes the structural elements that compose GLUE! from a high-level perspective. Besides, their integration contracts, and their individual expected behaviors are also detailed here. This coarse-grained description can be supplemented with additional information about lower level details, which are later explained in the implementation chapter.

3.4.1. Overview of the architecture

Figure 3.1 shows an overview of the GLUE! architecture. GLUE! follows a *three-tier architecture*[1] with **loosely-coupled distributed services**, where m VLE and n tool contracts are adapted through an intermediate software layer and a set of adapters. Therefore, GLUE! follows the second architectural approach presented in section 2.5.3 for the integration of external tools in VLEs. This architectural design aims at reducing the development effort, since the required integration code is partially assumed by this common intermediate software layer.

[1] It is important not to mistake this three-tier architecture, which is composed by three kinds of independent software elements, for the three-tier client-server architecture mentioned in chapter 2 (and followed by most VLEs and tools), in which each single software element presents three different layers: presentation, business logic and data management. Actually, each individual element in GLUE! may also present different layers, as it will be later exemplified in the implementation chapter.

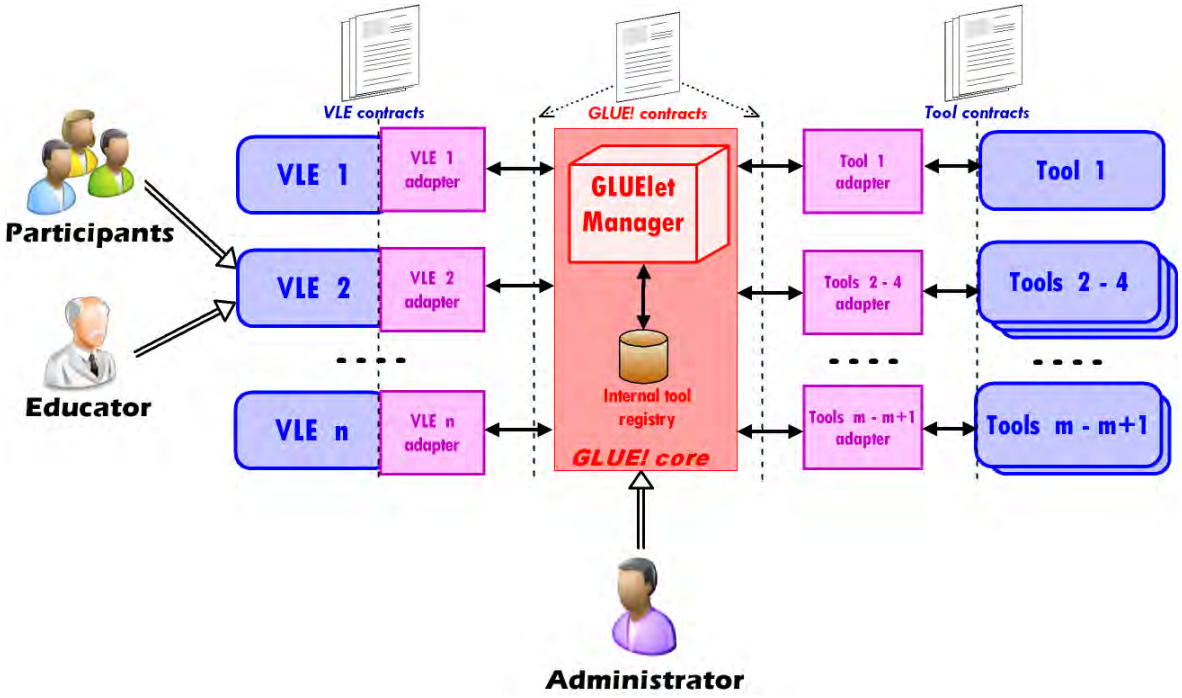

Figure 3.1: Overview of the GLUE! architecture.

The leftmost and rightmost GLUE! tiers make use of the well-known **adapter pattern** [Gam95] to respectively wrap VLEs and tools, also adapting their specific and heterogeneous contracts to two generic and homogeneous intermediate contracts: the *GLUE! integration contract for tools*, and the *GLUE! integration contract for VLEs*; these contracts are detailed in sections 3.4.2 and 3.4.3. These type of software adapters, which were also employed in some recent two-tier research integration works (e.g. IMS LTI or Basic LTI) to enable the integration of VLEs and tools **without modifying their implementations**, are called *VLE adapters* and *tool adapters* in the GLUE! architecture.

The three-tier architecture also contains an intermediate software layer, called *GLUE! core*, which offers the GLUE! integration contracts. The purpose of the GLUE! core is to decouple VLE and tool adapters, thus facilitating their independent development, while assuming most of the integration functionality, thus reducing the development effort. Besides, the GLUE! core acts as a bridge that connects heterogeneous VLEs and tools managing also the security issues, which are explained in section 3.6. VLEs are connected to the GLUE! core through VLE adapters that meet the GLUE! integration contract for VLEs, while tools are connected to the GLUE! core through tool adapters that comply with the GLUE! integration contract for tools. It is noteworthy that the GLUE! core promotes a **many-to-many integration**, since every new tool adapter developed for a tool enables its integration in any VLE with its corresponding VLE adapter, and the other way around. This is a major advantage of the GLUE! architecture.

The GLUE! architecture provides the functionality to create, configure, retrieve, update, and delete external tool instances, thus **managing the life cycle of external tools**, which is common to many software tools, including most VLE built-in tools, as discussed in chapter 2. This life cycle can be combined with the VLE features for the management of groups and activities, in order to associate each external tool instance to each group that participates in a given activity. Thereby, the students belonging to the same group may collaborate, sharing the same instance, as they normally do with VLE built-in tools. By using this functionality practitioners are greatly aided in the instantiation and enactment of individual and collaborative activities.

In order to support the management of the tool life cycle within VLEs, each of the three tiers in the GLUE! architecture has a clear role. Requests to create, configure, retrieve, update or delete tool instances are initiated in the VLE user interface; VLE adapters must thus process and send these request to the GLUE! core. The GLUE! core routes them to the corresponding tool adapters, which receive and process these requests, resulting in invocation on tool providers in tool specific ways. The following paragraphs deepen into the responsibilities and challenges of these three tiers.

The GLUE! core includes a processing element, called *GLUElet Manager*, which acts as a broker, receiving requests related to the support of the tool life cycle from VLE adapters (e.g "create two instances of tool T with configurations $C1$ and $C2$ that will be shared among students $S1-S3$ and $S4-S6$, respectively"), redirecting them to the appropriate tool adapters, and persisting information on the created instances. The GLUElet Manager, must thus be aware of the external tools available for integration, and how to reach the tool adapters that wrap them. This information is kept in the *internal tool registry*, which is also part of the GLUE! core. Significantly, the architectures that do not include intermediate software layers require VLE and tool adapters to assume these responsibilities. The fact that these responsibilities are placed in the GLUE! core reduces the functionality to be provided by the adapters, thus simplifying their development.

Tool adapters receive homogeneous requests from the GLUElet Manager for creating, configuring, retrieving, updating and deleting tool instances. These requests are adapted to comply with the contracts imposed by specific external tools. Interestingly, tool adapters are aware of the specific features and configurations supported by each tool, and so, they are responsible for providing configuration templates with the parameters that can be set during the process of creating external tool instances. These configuration templates are rendered as VLE-like forms by VLE adapters to be filled out by educators, and afterwards returned to tool adapters, which can use them, together with their knowledge of the specific tool contracts, to create instances with given configurations. The advantage of this approach is that if a tool provider adds interesting configuration properties, templates can easily be updated, without requiring any further

modifications in the architecture. Finally, it should be noted that if a number of tools have a similar contract and a similar collection of configuration parameters, it may be worthy to develop a single tool adapter to communicate with them all, instead of several adapters with minimal differences. An example of such implementation decision will be provided in next chapter.

The remaining tier of the architecture is formed by VLE adapters, where the requests of the tool life cycle are actually started. In fact, these requests are made by end-users on the VLE graphical interface. Thus, VLE adapters should capture and forward them to the GLUElet Manager. In addition, VLE adapters are responsible for the assignment of external tool instances to VLE users. To do so, VLE adapters should map the users, groups, and activities defined in VLEs to the instances that are actually created in external tools (i.e. those users belonging to the same group in an activity share the same external tool instance), as explained during the study of the tool life cycle. Unlike in the case of tools, VLEs rarely present similar contracts, and so, a specific VLE adapter is usually required for each of them.

Table 3.2 summarizes the purpose and functionality of each of the three GLUE! tiers: VLE adapters, the GLUE! core and tool adapters. Different providers can independently offer the elements in the three GLUE! tiers, except for VLE adapters, which are normally deployed together with VLEs. The reason is that VLEs adapters need to be implemented, in most cases, as VLE extensions, embedding partially their functionality in the VLE user interface.

Table 3.2: Elements in the GLUE! architecture: purpose and functionality.

Tier	Purpose	Functionality
GLUE! core	Promote a many-to-many integration Homogenize VLE and tool contracts Simplify the development of adapters	Manage requests related to the life cycle of external tools Manage persistent data about the available tools Manage security issues
VLE adapters	Connect VLEs and the GLUE! core	Enable the management and use of external tools within VLEs Map users, groups, and activities to tool instances
Tool adapters	Connect the GLUE! core and tools	Translate requests from the GLUElet Manager to tool contracts Provide and process configuration information

Finally, it is noteworthy to mention that the GLUE! architecture supports three roles, as depicted in Figure 3.1. The *administrator* of a GLUE! installation is responsible for populating the internal tool registry with information about the available external tools and their corresponding adapters. Besides, two other roles generate different actions within VLEs concerning the tool life cycle. The first role, namely the *educator* role, is an active role, since it generates actions that normally require interactions with external tools (creation, configuration, deletion and update of tool instances). This role is normally played by educators, but it can also be played by monitors, instructional designers, and any other user with special permissions to instantiate

learning designs within VLEs. The second role, namely the *participant* role, is a passive role, in the sense that it generates actions that do not require a GLUE-mediated communication with external tools, such as the retrieval of created instances. This role is normally played by students, although any VLE user participating in the enactment of a collaborative learning situation may play it.

3.4.2. GLUE! integration contract for tools

The GLUE! integration contract for tools is designed following some global criteria that stem from the decisions made in the design of the GLUE! architecture (see section 3.3). First, the *least number of restrictions* a contract has, the most likely it is adopted. Second, selecting *broadly accepted technologies* also facilitates adoption. Third, the contract should prescribe the *minimum functionality to support the life cycle of external tools, in order to minimize the development effort.* To facilitate its comprehension, the information regarding the GLUE! integration contract for tools is organized in three blocks, which correspond to each of these criteria: the restrictions on tool providers, the technological requirements on tool adapters, and the behavior expected from tool adapters.

Restrictions on tool providers

The GLUE! integration contract for tools must overcome two limitations found in the literature regarding the eligibility of external tools: the high number of restrictions that these tools must meet, and the low degree of adoption of such restrictions. According to the design decisions presented in section 3.3, the GLUE! integration contract for tools must impose few restrictions with a widespread adoption.

Actually, the GLUE! integration contract for tools imposes just one mandatory restriction that external tools must meet. The restriction is that **the code that external tools provide to enable the access to their functionality and data must be suitable for its distribution as a web content**. Web applications obviously meet this restriction; however, some standalone applications, such as those developed in Java whose code can be distributed in JNLP (Java Network Launching Protocol) [Jav11] files, may meet this restriction too.

In addition, this integration contract also reports one optional restriction that tools should meet in order to better exploit the GLUE! architecture. This restriction is that *tools should offer a programming interface* that allows the creation of tool instances, with different configurations, and for different users. If so, particularized tool instances for each group performing a collaborative activity may be provided, and without modifying the code of external tools. Otherwise, all the students in all the learning activities will access to the same functionality and data in

the external tool, as it happens in Basic LTI. Besides, the GLUE! contract for tools recommends this interface to be public, so that external developers may contribute to the development of tool adapters. Interestingly, the GLUE! contract for tools does not define any additional restrictions about the kind of interfaces, nor the technologies, nor the methods that must be offered by external tools.

Table 3.3 illustrates the compliance to these restrictions with representative examples. The seven most outstanding tools currently used in education meet the mandatory restriction, and so, they can be integrated in VLEs through the GLUE! architecture. Besides, there is high compliance regarding the optional restriction, and only Prezi does not present a programming interface. Nevertheless, multiple online references can be found in forums about Prezi users claiming for a programming interface. This public claim may encourage Prezi staff to prioritize the development of this interface in a short time. Moreover, the 69% of the Top 100 Tools for Learning[2] are distributed online, and so, they clearly meet the mandatory restriction, many of them complying with the optional restriction too. This number even grows if VLEs and hardware devices (e.g. Kindle[3]) are excluded from this list.

Table 3.3: Restrictions imposed on tools and degree of adoption. Mandatory restrictions are marked in bold, while optional restrictions are marked in italics.

Restriction	***Twitter***	***Youtube***	***Google Docs***	***Skype***	***Wordpress***	***Prezi***	***Slideshare***
Suitable for distribution as a web content	✔	✔	✔	✔	✔	✔	✔
Programming interface	✔	✔	✔	✔	✔	✘	✔

Technological requirements on tool adapters

The GLUE! integration contract for tools must overcome the limitations found in some related works regarding the high development effort required to integrate external tools in VLEs. To do so, the requirements chosen for the communication and development of tool adapters are based on popular and well-defined loosely-coupled technologies. The main advantage of employing loosely-coupled technologies regarding the development effort is the opportunity to easily reuse code from other existing components [Sur09], no matter their architectural approach or programming language. The main issues that should be considered regarding the communication and development of tool adapters are explained next, together with the technological requirements agreed. They are all are summarized in Table 3.4.

[2] `http://c4lpt.co.uk/top-tools/top-100-tools-for-learning-2011`. Last visited: June 2012.
[3] `http://kindle.amazon.com`. Last visited: June 2012.

Table 3.4: Technological requirements on tool adapters.

	Problem	*Technological requirement*
1	Remote invocation technologies and interfaces definition	**To be designed as REST services, offering a RESTful interface**
2	Requests and responses representation format	**To process requests and responses in the Atom format**
3	Configuration templates representation format	**To provide XForms or HTML5 configuration templates**
4	Instances naming and identification	**To provide URLs identifying ool instances**

1. *Remote invocation technologies and interfaces definition.* Tool adapters act like services to be invoked (at least) by the GLUElet Manager. Therefore, they must support some form of remote invocation, as well as publish the invocable interfaces. RPC-like middlewares, like Java RMI[4], CORBA [Vin97] or W3C Web Services [Cur02], have been popular approaches for remote invocation, but they suffer from drawbacks such as language dependency, promotion of tight coupling, or computational inefficiencies [Vin02].

   An alternative approach for tool adapters is a REST design. REST is a popular architectural style [Sha96a] defined by Roy Thomas Fielding in his thesis in 2000 [Fie00], to enable the simple communication among distributed elements and systems on the Web. According to Fielding's work the main restriction of REST-based systems is the uniform interface, which must consist of a set of fixed and well-defined methods. This restriction promotes the simplicity, scalability and easy development of applications on the Web. REST is not restricted to any transfer protocol, although HTTP (Hypertext Transfer Protocol) [NWG99] is usually selected as the communication protocol due to its popularity among web applications. Abstractions of any kind of information in REST are called *resources*, and are identified through URIs (Uniform Resource Identifiers) [NWG05b]. REST services (also called RESTful Services or RESTful Web Services) are a popular technology among web applications that combine the REST principles with client-server architectures [Ric07]. Resources exposed by REST services are also identified through URIs, being typically accessed through four well-defined HTTP methods: `POST` to create a new resource; `GET` to retrieve the information contained in a resource; `PUT` to update the information contained in a resource; and `DELETE` to delete a resource. Nevertheless, each REST service is responsible for defining the set of resources that are exposed, and the data exchange format in which the information is transmitted and stored.

   Designing tool adapters as REST services enables a low degree of coupling that eases their quick and independent development, as compared to other tighter service-oriented alternatives [Pau08], such as W3C Web Services, and of course, as compared to other even tighter non-service-oriented alternatives, such as Java RMI or CORBA. Besides, the definition

[4] `http://docs.oracle.com/javase/1.4.2/docs/guide/rmi/spec/rmiTOC.html`. Last visited: June 2012.

of RESTful interfaces in tool adapters facilitates the implementation of the CRUD-like (create, read, update and delete) [Mar83] methods in charge of the tool life cycle, as it is later shown in this section. Finally, it is important to remark that many tools and web applications are currently offering REST-based contracts, as it was studied in the feature analysis of software tools in chapter 2, thus facilitating also their wrapping by tool adapters. All in all, designing tool adapters as REST services facilitates their interoperability with external tools, and reduces their development effort.

Therefore, **tool adapters must be developed as REST services providing a RESTful interface to be invoked by the GLUElet Manager**. This interface must offer a set of resources identified by URIs and accessed through uniform interfaces. The specific resources that must be provided by tool adapters are presented later in this section.

2. *Requests and responses representation format.* RPC-like middleware and Web Services normally impose some type of representation for requests and responses (e.g. Java serialized objects, SOAP envelopes). Nevertheless, the REST style does not impose any specific data exchange format. In order to achieve interoperability among the elements of the GLUE! architecture some common representation format must be agreed. Here, the Atom Syndication format [NWG05a] is chosen as the format that tool adapters must support for the exchange of data, mainly due to its native use by many web applications, such as blogs, forums, or podcasts [Ric10].

   Atom is an open, extensible, XML-based language that is commonly used for publishing and editing information in the Web. Atom is also the format used for the transmission of information in the form of *web feeds* within many web sites and web tools [Say05]. Users may subscribe to these web sites and web tools and receive feeds, which are aggregated through feed readers, like Google News[5]. The information in each of these feeds is composed by a number of *entries*, which may contain headlines, articles, and links to other contents. Besides, every feed (and every entry) must attach some extra metadata including an author, an identifier, a title and a timestamp. Therefore, Atom is a well-known format that can be used to send and receive information among applications developed following web technologies.

   Other data formats like XML or RSS[6] (Really Simple Syndication) could have been chosen instead of Atom, also due to their popularity among web applications. Nonetheless, Atom is a more specific solution for this context because it extends XML for the specific purpose of transmitting web feeds. On the other hand, RSS was discarded because, unlike Atom, it is not an official specification. Furthermore, few improvements have been done on RSS in

[5] `http://news.google.com`. Last visited: June 2012.
[6] `http://rssboard.org/rss-2-0-8`. Last visited: June 2012.

the last few years, and the number of tools currently using RSS is much lower than those using Atom.

Therefore, **tool adapters must be prepared to process requests and responses in the Atom format**. Actually, the data format defined in the GLUE! integration contract for tools is a specialization of the Atom data format, where some minor elements needed by the architecture, such as the tool provider or the tool type, are added. These extra elements are defined in a specific *namespace* (`http://gsic.uva.es/glue/1.0`), as it is recommended by the Atom extension mechanism. Examples with the Atom requests and responses for tool adapters, including the elements needed by the architecture, can be consulted in the documentation of the GLUE! data format, which is available in the Appendix B of this dissertation.

3. *Configuration templates representation format.* Tool adapters are aware of the specific features and configurations supported by the tools they wrap. So, they must provide configuration templates, which must be shown to educators (typically embedded in the VLE graphical interface) when creating and configuring instances. In order to facilitate the processing of these templates in the VLE side, it would be convenient to have them formalized in a language that could be interpreted by the browser itself, since VLEs are normally web platforms; both XForms [W3C09a] and HTML5 [W3C12] potentially meet this demand.

   XForms is a W3C standard for the creation and management of forms for the Web. This standard has several important advantages compared to traditional forms in HTML documents[7]. First, XForms is an XML application, and so, it can be included as part of other XML-based documents like XHTML or Atom. Besides, XForms clearly splits the data and the representation of the data by using a model-view-controller approach [Kra88]. Therefore, input values for the parameters defined in an XForms document can be validated by the client of the application, avoiding unnecessary round-trips to the server. Furthermore, browsers or web clients may easily represent data in their graphical user interfaces with no or little processing. Nevertheless, no widely used browser supports natively XForms at the time of writing, and only a special *add-on* for XForms is compatible with some Firefox versions.

   HTML5, the fifth major revision of the HTML standard, is a well-known W3C specification for structuring and presenting content in the Web. HTML5 promotes the creation and management of web forms by incorporating much of the functionality of the superseded Web Forms 2.0 [W3C09b]. HTML5 has been quickly adopted among web applications in the last months, mainly led by social networking sites and search engines. Besides, HTML5

[7] `http://w3.org/TR/html4/interact/forms`. Last visited: June 2012.

is currently supported by several browsers, including Google Chrome, Firefox or Safari. Interestingly, HTML5 documents can be serialized as XML documents using XHTML5, being this way easily included as part of other XML-based documents like Atom.

Therefore, **tool adapters must be able to provide configuration templates in XForms or HTML5**, so that educators may set the values for those parameters that can be configured in external tools when creating instances. The GLUE! integration contract for tools currently recommends both representation, although, in the future it might recommend only HTML5 or XForms, depending on their degree of adoption.

4. *Instances naming and identification.* Once instances are created, unique identifiers must be provided by tool adapters to the outside world. A classical approach has been the generation of *ad hoc* alphanumeric tags, or pseudo-random strings. Nevertheless, here, a URL (Uniform Resource Locator) [NWG05b] representation has been chosen, so that external tool instances can be uniquely named and identified, but also easily embedded in web-based VLEs, thus highly simplifying the graphical integration of external tools.

   URL refers to a subset of URI that, apart from identifying a resource, provides a means to locate it. URLs are commonly used to access to resources and contents via web browsers. The prefix of the URL (e.g. `http`) determines how the URL must be interpreted by the web browser. Web browsers can represent different kinds of resources to end-users, depending on the URL (web sites, files, web application clients, etc.). Besides, these resources may contain hyperlinks to other resources, so that users can navigate through them.

   Therefore, **tool adapters must be able to provide URLs representing tool instances**. This design decision is consistent with the treatment of other resources in tool adapters, which are represented as URIs, as imposed by the REST-based contract. Significantly, if an external tool is not a web application, then the tool adapter must wrap it to provide a URL. However, many SaaS and Web 2.0 tools are offered as web tools, and so, in most cases, tool adapters do not need any further processing concerning this task.

Behavior expected from tool adapters

The GLUE! integration contract for tools is designed to impose few restrictions with a widespread adoption, and to support a low degree of coupling in order to reduce the development effort. These design decisions limit the degree of functionality that can be offered, as discussed in the previous chapter. Despite this limitation, the overall GLUE! architecture is designed to support the management of the tool life cycle, thus facilitating the instantiation and enactment of collaborative learning situations, due to their importance in the learning process [Dil99] and in the pedagogical models of VLEs and software tools [Bow11]. Tool adapters are expected to support this life cycle for the tools they wrap, by offering the following behavior:

1. The retrieval of the configuration template for one external tool.
2. The creation of external tool instances with some given configuration values, and shared among a list of end-users.
3. The retrieval of external tool instances, once created.
4. The update of the list of end-users sharing external tool instances, once created.
5. The deletion of external tool instances, once created.

This is the minimum functionality that must be offered by tool adapters according to the GLUE! integration contract for tools. The GLUElet Manager employs this functionality to integrate external tools in VLEs. Significantly, those interested in the development of tool adapters might extend this behavior adding new functionality to tool adapters, as long as they meet the GLUE! contract for tools and the contracts imposed by the own external tools. The internal implementation of this functionality is a responsibility of those developing tool adapters. Nevertheless, in order to facilitate this development, some guidelines about the actions resulting into this behavior are advised (although they are not imposed). Table 3.5 presents the expected behavior and the coarse-grained actions advised for tool adapters. Besides, this table summarizes the REST resources and HTTP methods that must be offered by tool adapters, as well as the information expected in HTTP requests and the responses that must be returned in case of success.

Some important clarifications must be done on this summary table. First, since tool adapters manage two kinds of information (generic information regarding the configurations of the tools wrapped, and specific information regarding the tool instances created), then two different kinds of REST resources are considered (*configuration* and *instance*). These resources are typically exposed following a format like `http://host:port/ToolAdapter/resource_name`.

These REST resources support some of the four CRUD methods reported in the HTTP specification [NWG99], as detailed in Table 3.5: `POST` for the *creation* of a new resource with a given state; `GET` for the *retrieval* of the state (or part of it) of a resource; `PUT` for the *update* of a resource, by completely substituting its state; and `DELETE` for the *deletion* of the resources and its state. It is convenient to note that the `/instance/{instanceId}` resources support a `POST` method, rather than a `PUT` method, for the update of the list of end-users sharing instances, since only a part of the state of these resource is modified. The remaining resources and methods need no further explanation. The REST interface designed for tool adapters follows the guiding principles defined in [Fie00].

HTTP `POST` requests can include information in the request body (Atom feeds in this case) [NWG99]. Nevertheless, the request body should be avoided in `GET` and `DELETE` requests,

Table 3.5: Behavior expected from tool adapters: resources and methods that must be offered by tool adapters to be invoked by the GLUElet Manager; coarse-grained actions advised and expected response from tool adapters.

	REST resource offered	**HTTP Method offered**	**Information included in the HTTP request**	**Expected response**	**Coarse-grained actions advised**
1	`/configuration`	`GET`	"*tool*" attached to the URI of the request	Configuration template	Search the configuration template for that tool. Return the configuration template.
2	`/instance`	`POST`	Atom containing: "*tool*", "*configuration*", "*users*", "*callerUser*", "*parameters*"	URI of the `/instance/{instanceId}` resource. Optionally, any extra information required for the update or deletion (e.g. a different URI)	Create a new resource `/instance/{instanceId}`. Create and configure a new external tool instance. Represent the tool instance in a URL. Store the URL in the `/instance/{instanceId}` resource. Return the URI of that resource.
3	`/instance/{instanceId}`	`GET`	"*callerUser*" attached to the URI of the request	URL representing an external tool instance	Search the instance for that user. Return the URL for that instance
4	`/instance/{instanceId}`	`POST`	Atom containing: "*tool*", "*users*", "*callerUser*", "*parameters*"	URI of the `/instance/{instanceId}` resource	Update the list of users sharing the same instance. If needed, update the external tool instance. Return the URI of that resource.
5	`/instance/{instanceId}`	`DELETE`	"*callerUser*" and "*parameters*" attached to the URI of the request	Ok	Delete the `/instance/{instanceId}` resource. Delete the external tool instance. Return Ok.

and so, the data that need to be sent (see Table 3.5) is attached in the URL, as it is recommended when designing REST-based applications [Ric07]. Responses do not find this problem and they all can be formatted in Atom.

There are some information contained in HTTP requests that are intended to be used either by tool adapters or by external tools. One example is the "*configuration*" filled out by the educator, which is included in those requests aimed at creating external tool instances. Another examples are the list of users ("*users*") sharing instances, and the particular "*tool*"; both are included in the creation and update requests, being the latter also employed to retrieve the configuration template. Besides, the name of the end-user that makes requests ("*callerUser*") is attached in most messages as a means of identification with the elements of the architecture and with external tools. Besides, there are some special information included by the GLUElet Manager in a generic field called "*parameters*". Examples of the information contained in this field are the tool provider or credentials for the authentication with external tools. Only those requests that may need to establish a communication with the tool (creation, update, deletion) include this special field.

3.4.3. GLUE! integration contract for VLEs

The GLUE! integration contract for VLEs is also designed following the three global criteria presented in the previous section: imposing the least number of restrictions, selecting broadly accepted technologies, and prescribing the minimum functionality to support the life cycle of external tools. This contract is explained following the same structure as that followed in the case of tools.

Restrictions on VLE providers

The GLUE! integration contract for VLEs must overcome the two main limitations found in the literature regarding the eligibility of VLEs: the high number of strict restrictions that VLEs must meet, and the low degree of adoption of such restrictions. Therefore, and according to 3.3, the GLUE! integration contract for VLEs must impose few restrictions with a widespread adoption.

Actually, the GLUE! integration contract for VLEs only imposes three mandatory restrictions that VLEs must meet. First, **VLEs must be able to render web contents**, so that tool instances can be easily embedded in VLEs, as *IFrames*, or HTML *Objects* [W3C12]. This restriction stems from the fact that most existing tools are currently developed as web tools, and thus, imposing this restriction highly simplifies the code required for the graphical integration of external tools. Main VLEs meet this restriction, since they are all web-based platforms. Second, **VLEs must offer an extension interface**, so that VLE adapters can wrap VLEs, taking advantage of the functionality they offer in order to integrate external tools, but without modifying their code. Furthermore, if this interface is public, then external developers may contribute to the implementation of the corresponding VLE adapter. Most VLEs meet this restriction, since they have been typically conceived to promote the extension of their functionality; however, the extension interface is not always public (e.g. Blackboard). Third, **VLEs must understand the concept of tool or a similar one**. For example, in the case of Moodle or LAMS the concept of tool is mapped to that of activity, while others such as Blackboard or Sharepoint LMS explicitly use the term tool. This restriction is essential, because otherwise, the integration of external tools would not make sense. Main VLEs also comply with this restriction, being tools a key concept in their behavior.

Besides, the GLUE! integration contract for VLEs reports two optional restrictions that VLEs should meet to fully exploit the features offered by the GLUE! architecture. First, *VLEs should understand the concept of group*, so that external tool instances can be shared among the students that belong to the same groups. Otherwise, collaboration and groupwork would not effectively be achieved through VLEs. Besides, *VLEs should understand the concept of role*, so

that the roles defined by GLUE! can be mapped to the roles defined by VLEs. If not, any VLE user could participate in the instantiation of learning designs, something that does not fit within the educator-centered pedagogical model of VLEs.

Table 3.6 shows the compliance to these restrictions by main VLEs. All of them meet both the mandatory and optional restrictions, since these restrictions come from a previous analysis of their features.

Table 3.6: Restrictions imposed on VLEs and degree of adoption. Mandatory restrictions are marked in bold, while optional restrictions are marked in italics.

Restrictions	***Moodle***	***LAMS***	***.LRN***	***Sakai***	***Blackboard***	***Claroline***	***SharePoint LMS***
Render web contents	✔	✔	✔	✔	✔	✔	✔
Extension interface	✔	✔	✔	✔	✔	✔	✔
Concept of tool	✔	✔	✔	✔	✔	✔	✔
Concept of role	✔	✔	✔	✔	✔	✔	✔
Concept of group	✔	✔	✔	✔	✔	✔	✔

Technological requirements on VLE adapters

The GLUE! integration contract for VLEs must also overcome the limitations found in some related works regarding the high development effort required to integrate external tools in VLEs. To overcome these limitations, the GLUE! integration contract for VLEs follows the same approach as in the case of tools, being the requirements chosen for the communication and development of VLE adapters based on popular and well-defined loosely-coupled technologies. Actually, the same technological requirements have been agreed to promote the interoperability among the elements of the architecture. The main issues regarding the communication and development of VLE adapters, as well as the technological requirements agreed, are explained next, being summarized in Table 3.7.

Table 3.7: Technological requirements on VLE adapters.

	Problem	*Technological requirement*
1	Remote invocation technologies and interfaces definition	**To invoke a RESTful interface**
2	Requests and responses representation format	**To process requests and responses in the Atom format**
3	Configuration templates representation format	**To process XForms or HTML5 configuration templates**

1. *Remote invocation technologies and interfaces definition.* VLE adapters invoke the GLUElet Manager in order to request the creation, configuration, retrieval, update and deletion of tool instances. These requests must be sent using some form of remote invocation. Here, it is important to put forward that the GLUElet Manager has been designed as a REST service, aimed at promoting a loosely-coupled integration, and facilitating the implementation of the CRUD-like methods in charge of the tool life cycle, as later discussed in section 3.4.4.

   Therefore, **VLE adapters must be able to invoke the RESTful interface offered by the GLUElet Manager**. The specific resources and methods that can be invoked by VLE adapters are presented later in this section. It is noteworthy that VLE adapters do not need to be implemented as REST services, nor to offer RESTful interfaces. The reason is that VLE adapters are not designed as services for other elements of the architecture; they just extend the behavior of VLEs and communicate with them through internal processes. Significantly, the GLUE! integration contract for VLEs does not impose the definition of any interfaces on VLEs adapters. This clarification is pertinent, since most of the requirements for the development of VLE adapters come from VLE contracts and not from the GLUE! contract for VLEs.

2. *Requests and responses representation format.* The REST interface that must invoked by VLE adapters does not impose any specific data exchange format. Here, Atom is also agreed as the format in which VLE adapters must exchange information with the GLUElet Manager, due to the same reasons that were discussed in the previous section. Besides, this decision is consistent with the selection of Atom for the communication between the GLUElet Manager and tool adapters. Significantly, those requests or responses that the GLUElet Manager do not need to process can be redirected to VLE or tool adapters, since they all share the same data exchange format.

   Therefore, **VLE adapters must be prepared to process requests and responses in the Atom format**. More specifically, they must use the aforementioned specialization of the Atom format for the GLUE! architecture. Examples with these requests and responses for VLE adapters can be consulted in the documentation of the GLUE! data format, which is available in the Appendix B of this dissertation.

3. *Configuration templates representation format.* VLE adapters must show educators the configuration templates, so they can set the values for each external tool instance. Significantly, these configuration templates can be formatted in XForms or HTML5 files, which are two well-known specifications supported by some web browsers. Therefore, **VLE adapters must be able to process XForms or HTML5 templates** in those cases where web browsers cannot directly process the content of these configuration templates.

Behavior expected by VLE adapters

The GLUE! integration contract for VLEs is also designed to impose few restrictions with a widespread adoption, and to support a low degree of coupling in order to reduce the development effort. Therefore, and due to these premises, the degree of functionality that the GLUElet Manager can offer to VLE adapters is quite limited. Nevertheless, this functionality is enough to support the management of external tool instances, as well as the retrieval of information about the available external tools. VLE adapters may then expect the GLUElet Manager to expose the needed functionality to satisfy the following demands:

1. The retrieval of the list of available external tools.
2. The retrieval of functional information about one specific external tool.
3. The retrieval of the configuration template for one external tool.
4. The creation of external tool instances with some given configuration values, and shared among a list of end-users.
5. The retrieval of external tool instances, once created.
6. The update of the list of end-users sharing external tool instances, once created.
7. The deletion of external tool instances, once created.

VLE adapters can integrate external tools in VLEs making use of this behavior. Significantly, those developers interested in the implementation of VLE adapters may design multiple use cases over this functionality, as long as they meet the GLUE! contract for VLEs and the contracts imposed by the own VLEs. Section 3.5 details the recommended use cases, although they are not imposed. Obviously, the internal implementation of VLE adapters is a responsibility of those developing these adapters. Table 3.8 presents the coarse-grained actions advised for VLE adapters. Besides, this table summarizes the REST resources and HTTP methods that VLE adapters should use, the information that should be included in HTTP requests, and the responses that should be returned by the GLUElet Manager in case of success. Noticeably, the same clarifications presented in section 3.4.2 for the equivalent table in the case of tool adapters can be applied here. In this case, two kinds of REST resources (*tool* and *instance*) are defined to manage the generic information about the available external tools, and the specific information concerning the tool instances that are created. It is noteworthy that Table 3.8 includes information about the GLUElet Manager that is further explained next. Thereby, it is recommended to check also section 3.4.4 for a better understanding.

Table 3.8: Behavior expected by VLE adapters. Resources and methods that are offered by the GLUElet Manager, and that can be invoked by VLE adapters; coarse-grained actions advised from VLE adapters in order to send each request.

	Coarse-grained actions advised	**REST resource invoked**	**HTTP method invoked**	**Information that is expected in the HTTP request**	**Returned response**
1	Request the list of available external tools.	`/tools`	`GET`		List of available external tools in the internal tool registry
2	Request additional functional information about a tool.	`/tools/{toolId}`	`GET`		Functional information about one specific external tool
3	Request the configuration template for a tool.	`/tools/{toolId}/configuration`	`GET`		Configuration template
4	Request the creation and configuration of a new tool instance.	`/instance`	`POST`	Atom containing: "*tool*", "*configuration*", "*users*", "*callerUser*"	URI of the `/instance/{instanceId}` resource, which contains a reference to the external tool instance, in the GLUElet Manager
5	Request an external tool instance.	`/instance/{instanceId}`	`GET`	"*callerUser*" attached to the URI of the request	URL representing an external tool instance
6	Request the update of users sharing an instance after a modification in a group configuration.	`/instance/{instanceId}`	`POST`	Atom containing: "*tool*", "*users*", "*callerUser*"	URI of the `/instance/{instanceId}` resource in the GLUElet Manager
7	Requests the deletion of an external tool instance.	`/instance/{instanceId}`	`DELETE`		Ok

3.4.4. Technologies and behavior of the GLUElet Manager

The GLUElet Manager is the processing element of the GLUE! core in the GLUE! architecture. It manages the requests and responses related to the life cycle of external tools. These requests come from VLE adapters (after end-users' actions in VLEs) and are propagated to tool adapters through the GLUElet Manager. Besides, this element also manages the responses from tool adapters, returning them to VLE adapters. The GLUElet Manager employs an internal tool registry to store persistent information about the available external tools, and about the available tool adapters, thus knowing where to send each request. Significantly, the GLUE! integration contract for VLEs and the GLUE! integration contract for tools are defined according to the behavior of the GLUElet Manager. The behavior of the GLUElet Manager, including the resources and methods offered to VLE adapters, the resources and methods invoked on tool adapters, and the coarse-grained actions internally performed, are summarized in Table 3.9. The clarifications explained in section 3.4.2 for the equivalent table in the case of tool adapters should also be considered here.

Table 3.9: Behavior of the GLUElet Manager. Resources and methods offered by the GLUElet Manager are expected to be invoked by VLE adapters. Resources and methods invoked by the GLUElet Manager are expected to be offered by tool adapters.

	REST resource offered	**HTTP method offered**	**Coarse-grained actions**	**REST resource invoked**	**HTTP method invoked**
1	`/tools`	`GET`	Search the list of available external tools in the internal tool registry. Return the list of available tools.		
2	`/tools/{toolId}`	`GET`	Search the information about one specific external tool in the internal tool registry. Return the information about that tool.		
3	`/tools/{toolId}/configuration`	`GET`	Search the tool adapter for that external tool in the internal tool registry. Propagate another `GET` to the `/configuration` resource in that tool adapter. Return the configuration template, after the response of the tool adapter.	`/configuration`	`GET`
4	`/instance`	`POST`	Create a new resource (`/instance/{instanceId}`). Search the tool adapter for that external tool in the internal tool registry. Propagate another `POST` to the `/instance` resource in that tool adapter. Add the *parameters* field in the Atom of the request. Store the URI returned by this adapter in the `/instance/{instanceId}` resource, after the response of the tool adapter. Return the URI of this resource.	`/instance`	`POST`
5	`/instance/{instanceId}`	`GET`	Propagate another `GET` to the `/instance/{instanceId}` resource in the tool adapter. Return the URL of the external tool instance, after the response of the tool adapter.	`/instance/{instanceId}`	`GET`
6	`/instance/{instanceId}`	`POST`	Propagate another `POST` to the `/instance/{instanceId}` resource in the tool adapter. Add the *parameters* field in the Atom of the request. Return the URI of the `/instance/{instanceId}` resource, after the response of the tool adapter.	`/instance/{instanceId}`	`POST`
7	`/instance/{instanceId}`	`DELETE`	Propagate another `DELETE` to the `/instance/{instanceId}` resource in the tool adapter. Attach the *parameters* field to that URI. Return Ok, after the response of the tool adapter.	`/instance/{instanceId}`	`DELETE`

Regarding technologies, the GLUElet Manager is designed as a REST service with a REST interface that follows the guidelines presented in [Fie00]. Therefore, the GLUElet Manager exposes a set of resources identified by URIs that may be used by VLE adapters to access to the information and functionality offered by this element. Besides, the GLUElet Manager uses a set of resources, also identified by URIs, that must be exposed by tool adapters. That design decision facilitates the propagation of requests along the architecture, since the GLUElet Manager does not need to make technological changes in these requests to propagate them.

Requests and responses to and from the GLUElet Manager are formatted in the specialization of the Atom data format that was previously characterized in the definition of the GLUE! integration contracts. It is noteworthy that Atom feeds from VLE adapters are propagated to

tool adapters with no (or minor) changes in many requests (e.g. adding the *parameters* field). Significantly, the GLUElet Manager does not need to process configuration templates, nor URLs, insofar as both are transparently propagated along the architecture.

3.5. Overall behavior of the architecture

The elements of the architecture were individually explained in the previous section, detailing the behavior offered by the GLUElet Manager, and the behavior that is expected to be offered by VLE and tool adapters. This section presents the overall behavior of the GLUE! architecture, exemplified through the typical use cases that may occur during the instantiation and enactment of collaborative learning situations that require the integration of external tools. Obviously, these use cases are oriented to support the tool life cycle within VLEs in its four phases: *creation, configuration and assignment of external tool instances*; *use of external tool instances*; *update of users sharing external tool instances*; and *deletion of external tool instances.* It is noteworthy that these are illustrative examples of the behavior of the architecture, and that new use cases (or modifications on these ones) may be made, as long as these new use cases meet the GLUE! integration contracts for VLEs and tools. Examples of use cases built on top of these can be seen in the implementation chapter.

3.5.1. Use case 1: creation, configuration and assignment of external tool instances

The first use case is the most complex of the four. It supports the functionality to create and configure external tool instances within VLEs, and to assign them to VLE users depending on the group they belong to. Figure 3.2 shows the sequence diagram of this use case. Someone playing a role that can carry out the instantiation of a collaborative learning situation in the VLE (most likely the educator or, in some settings, an instructional designer) may be the actor of this use case. This use case starts with the educator adding a tool to a new activity in his commonly used VLE. The educator can add a VLE built-in tool as usual, or an external tool (interaction 1.1). In the second case, the VLE adapter retrieves the list of available external tools from the GLUElet Manager through a `GET` to the `/tools` resource (1.2-1.4). Each tool listed includes some extra information (e.g. the tool provider) that may help the educator in their choice. After making a decision, the educator selects one of the available external tools (2.1). Then, the VLE adapter retrieves and render the configuration template for this tool, querying the GLUElet Manager through another `GET` to the `/tools/{toolId}/configuration` resource (2.2). This `GET` is propagated to the `/configuration` resource in the tool adapter (2.3), which

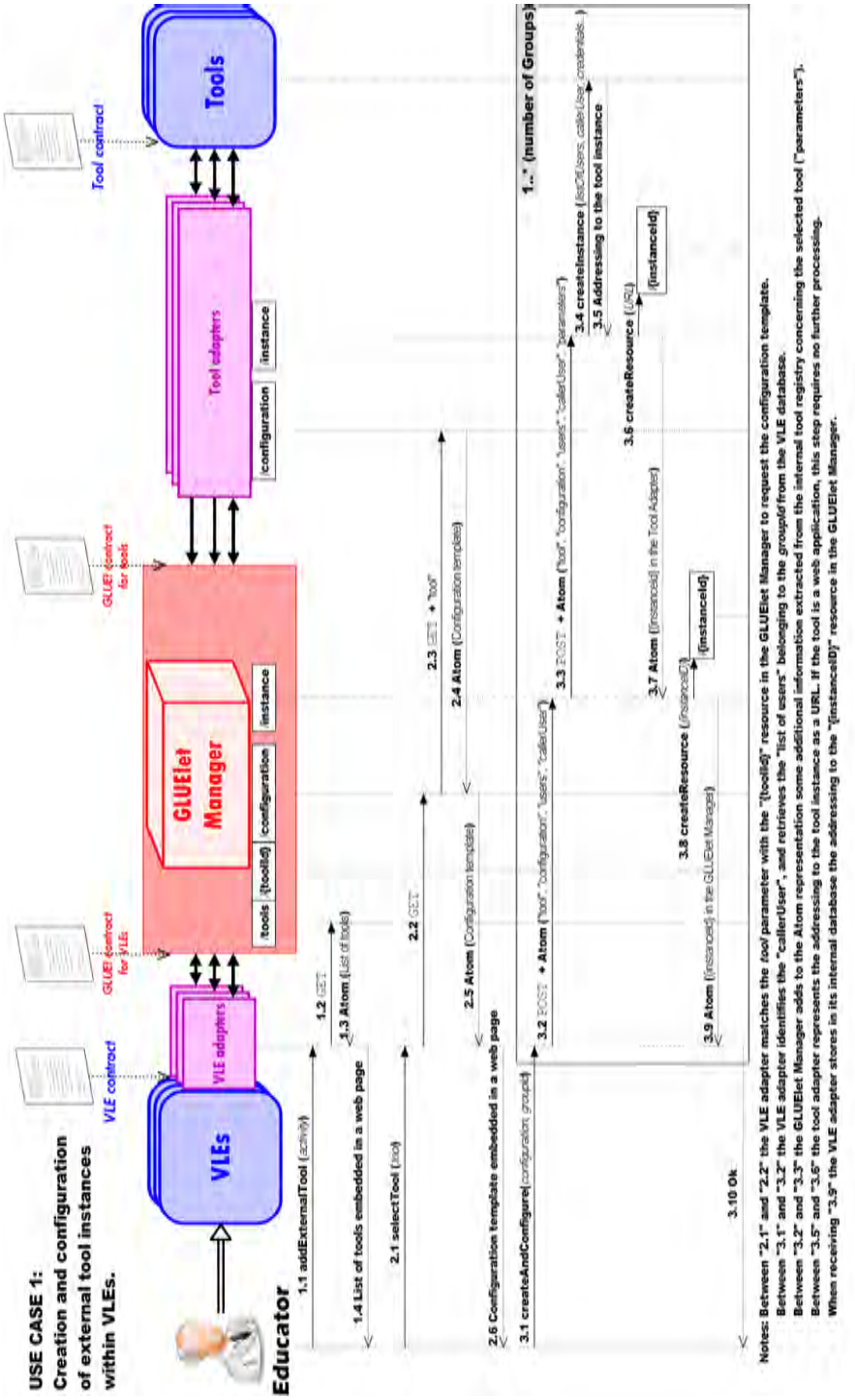

Figure 3.2: Sequence diagram representing the successful creation, configuration and assignment of external tool instances: interactions numbered `1.x` enable the retrieval of the list of available external tools; interactions numbered `2.x` enable the retrieval of the configuration template; interactions numbered `3.x` enable the creation, configuration and assignment of external tool instances. It should be noted that multiple instances (one for each group) can be transparently created in a single educators' request.

actually stores the configuration template for the selected tool, returning it back to the VLE adapter (2.4-2.6).

After that, the educator fills out the template, and requests the creation and configuration of tool instances for all the groups defined in that activity (3.1); in this step, the educator might also set different configurations for each group, requesting the creation of instances one by one. A `POST` to the `/instance` resource in the GLUElet Manager is sent for each of the groups with some additional data required to accomplish the creation of tool instances (3.2). This data includes the tool name ("*tool*"), the configuration filled out by the educator ("*configuration*"), the list of users belonging to the group the instance must be assigned to ("*users*"), and the VLE user that makes the request ("*callerUser*"). The `POST` is propagated by the GLUElet Manager to the corresponding tool adapter (3.3), including some extra parameters that may be required for the creation of the tool instance, such as the tool provider or credentials (in the "*parameters*" field). The tool adapter actually creates the instance as established in the tool contract (3.4-3.5), and generates a URL for end-users to access this instance (if the tool does not provide this URL). The URL is stored in a new resource (`/instance/{instanceId}`) to be later retrieved in the enactment phase (3.6). Finally, the successful response from the tool adapter to the GLUElet Manager includes the *instanceId*[8], which is stored (3.7-3.8), returning another local *instanceId* to the VLE adapter (3.9-3.10). Calls 3.2 to 3.9 are repeated for every group defined in the learning activity. Significantly, while educators explicitly create and configure external tool instances within VLEs, the assignment of these instances to VLE users is automatically made.

3.5.2. Use case 2: use of external tool instances

The second use case includes the functionality to retrieve and use external tool instances within VLEs. Figure 3.3 shows the sequence diagram of this use case. Once instances are created, those participating in the collaborative learning situation can retrieve, visualize and use them during the enactment phase. Though this use case is normally started by students or learners, educators could also retrieve, visualize and use external tool instances, in order to monitor students' work, give feedback to students, or mediate to facilitate the collaboration among participants. Therefore, external tool instances are shared among the members of a group, plus the educators or monitors supervising the activities, as it generally happens with VLE built-in tools.

This use case starts with the selection of an activity in which an external tool is intended to be used within the VLE interface (interaction 1.1). Then, the VLE adapter internally matches the activity, tool, and group identifier of the VLE user with an `/instance/{instanceId}` resource

[8]It should be noted that the values of *instanceId* are local and unique to the element generating them, but need not coincide with the values given in other elements of the GLUE! architecture.

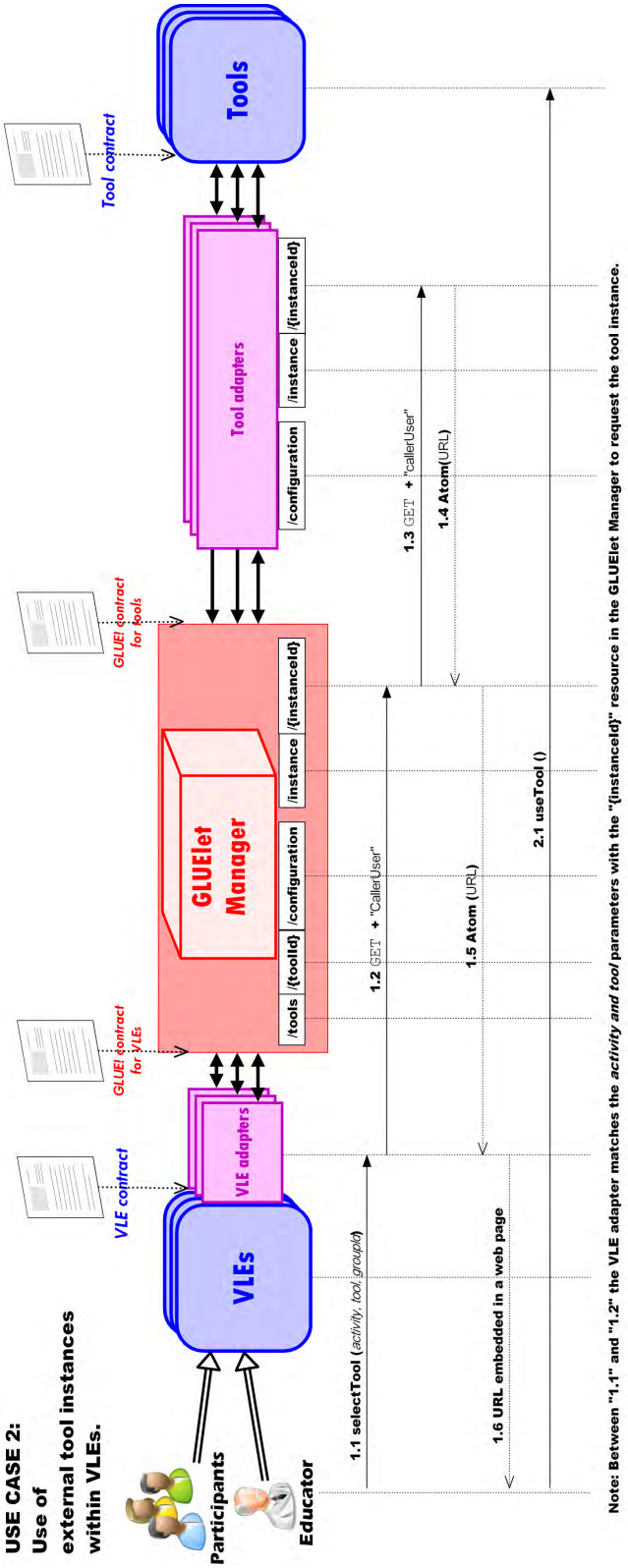

Figure 3.3: Sequence diagram representing the use of external tool instances.

in the GLUElet Manager. A `GET` request is then sent to this resource in the GLUElet Manager (1.2), which propagates another `GET` to the corresponding `/instance/{instanceId}` resource in the tool adapter (1.3). Both requests include the "*callerUser*" parameter attached to the URL, so that the external tool and the tool adapter may identify the user that requests the instance. After that, the tool adapter returns the URL representing the external tool instance (1.4-1.5), which is embedded in a web page within the VLE graphical interface (1.6). Finally, the participant can use the tool instance (2.1) in collaboration with other group members. It is noteworthy that there is no further communication between VLEs and external tools, once instances are retrieved.

3.5.3. Use case 3: update of users sharing external tool instances

The third use case includes the functionality to update the list of users sharing instances, as a result of group modifications within VLEs. Figure 3.4 presents the sequence diagram of this use case. Educators may decide to modify some of the groups after the instantiation of a learning design, for instance to react to common events that may occur in real educational scenarios (e.g. participants' absences or latecomers). Therefore, this use case can be considered useful to reconfigure the group structure and the components of each group. The GLUE! architecture enables the update of users, which, depending on the particular tool contract, may require an extra interaction with external tools. However, in some cases, tool adapters can manage the update of users without requiring an additional communication with external tools, as it is detailed with examples in the implementation chapter.

This use case starts with the educator modifying one or more groups within the VLE interface (interaction 1.1). The VLE adapter must be aware of these modifications through its internal notification mechanism with the VLE (1.2), so that the requests for the update of users can eventually start. The VLE adapters internally matches the groups that are modified and the instances affected (i.e. a set of `/instance/{instanceId}` resources in the GLUElet Manager). Next, and for each of these resources, a `POST` request is sent, including the new list of users ("*users*"), the tool name ("*tool*") and the caller user ("*callerUser*") (1.3), as it also happens in the creation use case. The GLUElet Manager propagates this `POST` to the equivalent `/instance/{instanceId}` resource in the tool adapter (1.4), but adding the parameters needed for the communication with the external tool (i.e. the "*parameters*" field). The tool adapter updates the list of users sharing that instance, sometimes through an additional interaction with the external tool (1.5-1.6), and sometimes by updating the information they store associated to that tool instance. The response to this process is the same as in the creation use case, in order to facilitate its implementation (1.7-1.8). There is a final confirmation to end-users after all the instances are updated (1.9-1.10).

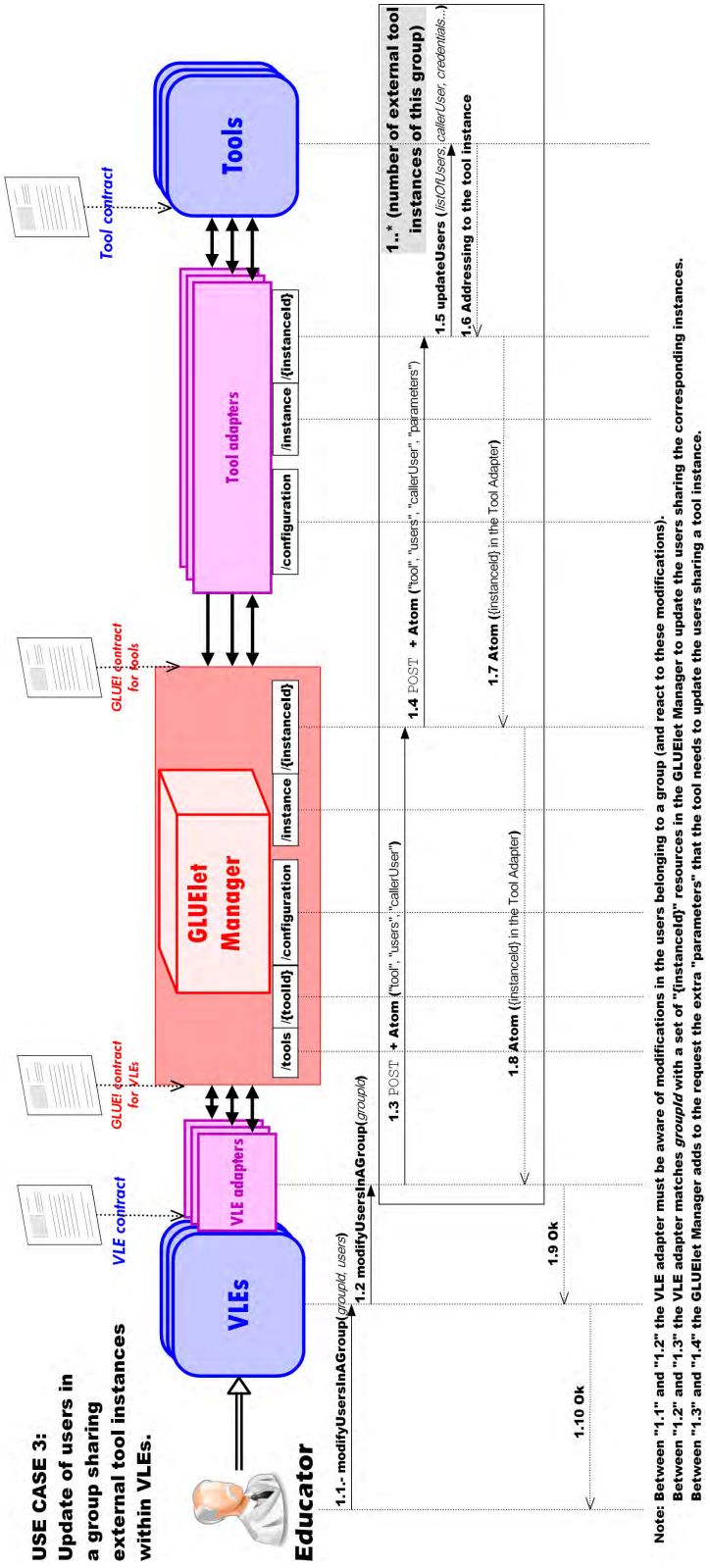

Figure 3.4: Sequence diagram representing the update of users sharing external tool instances.

3.5.4. Use case 4: deletion of external tool instances

The last use case includes the functionality to delete external tool instances within VLEs. Figure 3.5 depicts the sequence diagram of this use case. Educators may need to delete the external tool instances created for a learning activity, once this activity (or the entire collaborative learning situation) is finished. The GLUE! architecture enables the deletion of these instances at any moment after their creation, as well as the deletion of all the resources generated during the creation process.

This use case starts with the deletion of an activity or tool in a VLE (interaction 1.1). Then, the VLE adapter internally matches this activity or tool with one or more `/instance/{instanceId}` resources in the GLUElet Manager. Next, a `DELETE` request is sent to each resource (1.2) indicating who is the VLE user that makes the request ("*callerUser*"). For each request received by the GLUElet Manager, another `DELETE` request is propagated to the equivalent `/instance/{instanceId}` resource in the corresponding tool adapter (1.3), but adding the additional information ("*parameters*") that may be required to delete the external tool instance (e.g. the tool provider or credentials). The tool adapter actually deletes the external tool instance (1.4-1.5), and after that, the tool adapter and the GLUElet Manager delete the resources associated to that instance (1.6-1.7). This process is repeated for every group defined in the learning activity. Finally, the educator receives a confirmation of the finalization of the process (1.8).

3.6. Security issues

There are several security issues that should be taken into account when tackling the integration problem; these issues typically arise as a consequence of managing the data and resources belonging to integrated tools within external systems (VLEs). Besides, in this particular work, some additional security issues appear as a result of designing the GLUE! architecture as a set of distributed software elements, which may run on different premises, being communicated through computer networks. Besides, VLEs and tools are also involved in these issues, as usual. Figure 3.6 shows an overview of the six security issues detected in this work. Here, it is important to point out that dealing with these security issues is out of the main scope of this research, although solutions for all of them have been agreed. However, instead of proposing new approaches to tackle these issues, this dissertation relies on well-known mechanisms provided by VLEs, tools, or by other similar works in the literature.

1. *User level authorization for the use of VLEs.* VLEs normally allow the management of learning activities to a list of users that have been previously registered in that VLE. VLE

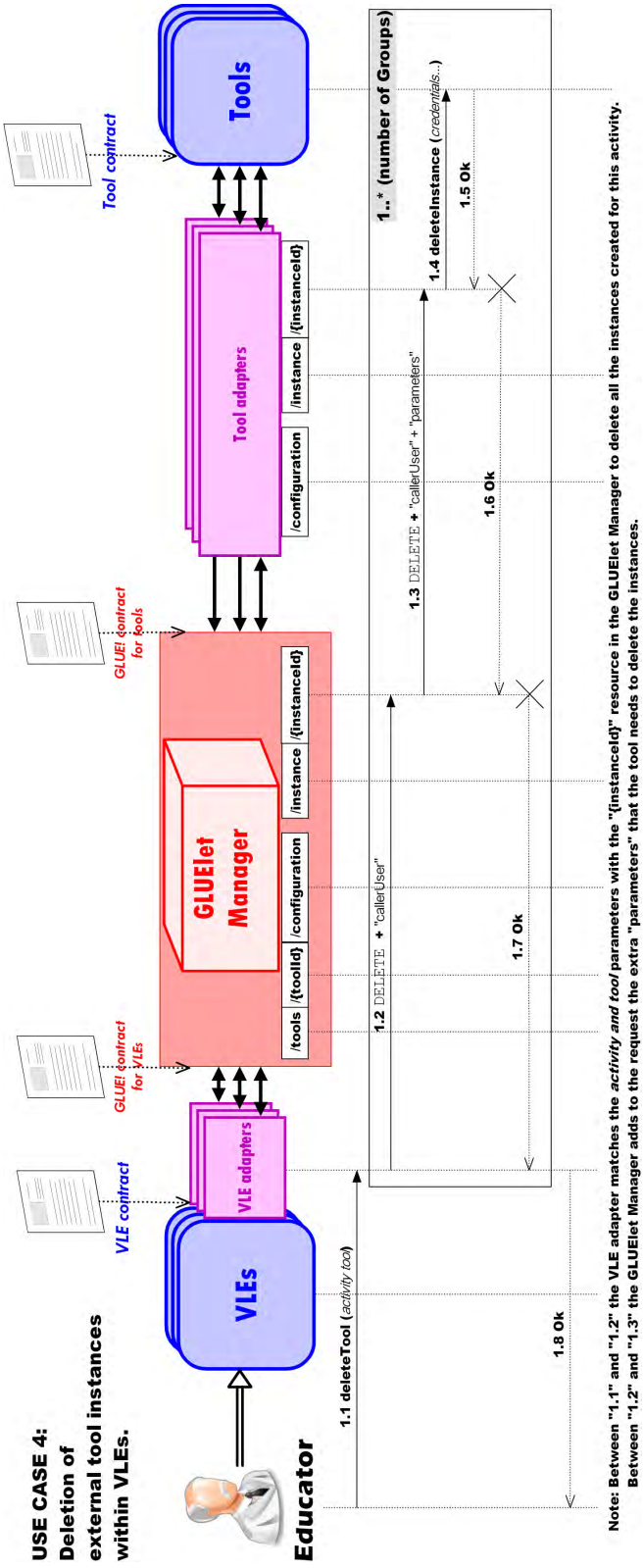

Figure 3.5: Sequence diagram representing the deletion of external tool instances.

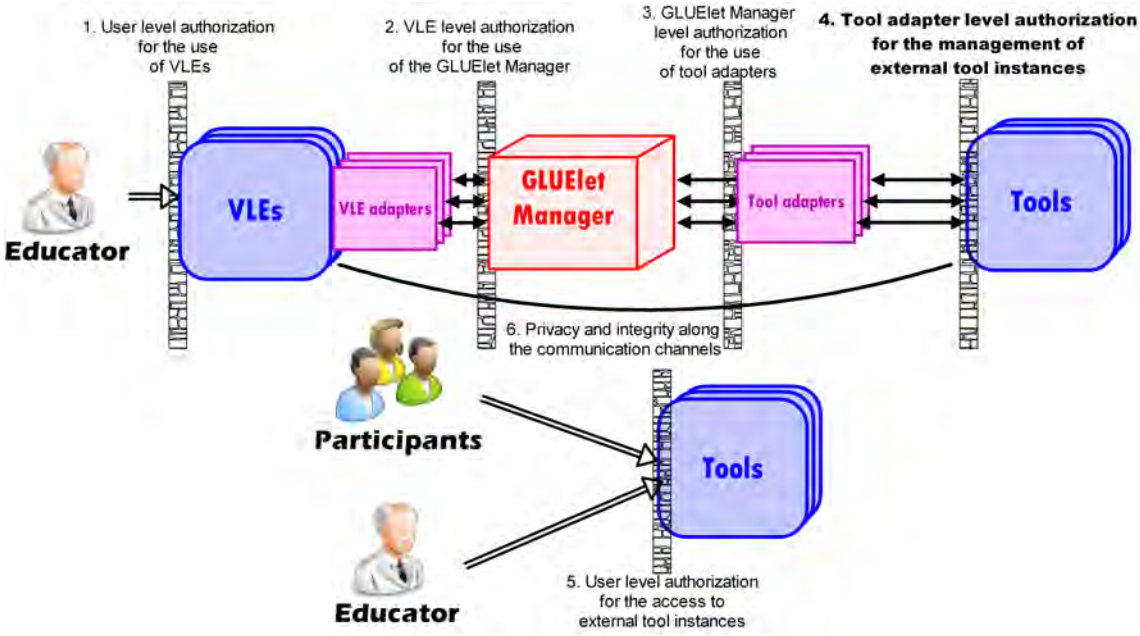

Figure 3.6: Overview of the security issues in the GLUE! architecture: 1. User level authorization for the use of VLEs; 2. VLE level authorization for the use of the GLUElet Manager; 3. GLUElet Manager level authorization for the use of tool adapters; 4. Tool adapter level authorization for the management of external tool instances; 5. User level authorization for the access to external tool instances; 6. Privacy and integrity along the communication channels

administrators are typically in charge of registering these end-users, assigning them different permissions (e.g. educators can instantiate learning activities, while students can enact them). In order to identify authorized users, VLEs normally implement some authentication mechanisms. For instance, Moodle reports more than fifteen ways for authentication[9]. Some of these mechanisms rely on internal or external servers, such as LDAP (Lightweight Directory Access Protocol) [How03] or CAS (Central Authentication Service) [Maz05] to get single-sign on (SSO), thus enabling end-users access to multiple platforms through a single authentication process. Besides, Moodle also supports SSO by building a federation with other systems, and using the Shibboleth infrastructure [Mor04]. Other VLEs, such as LAMS or Sakai, only report compatibility with LDAP[10][11] and Shibboleth[12][13], apart from the traditional manual authentication. All in all, this dissertation employs the native VLE mechanisms for the authentication and authorization of end-users, as a premise to use the GLUE! architecture.

[9] `http://docs.moodle.org/22/en/Authentication`. Last visited: June 2012.
[10] `http://wiki.lamsfoundation.org/display/lamsdocs/LDAP+Configuration`. Last visited: June 2012.
[11] `https://confluence.sakaiproject.org/display/DOC/LDAP+Integration`. Last visited: June 2012.
[12] `http://wiki.lamsfoundation.org/display/lams/LAMS+as+Shibboleth+IdP`. Last visited: June 2012.
[13] `http://devel.it.su.se/pub/jsp/polopoly.jsp?d=2376&a=21472`. Last visited June 2012.

2. *VLE level authorization for the use of the GLUElet Manager.* This issue arises when end-users call the GLUElet Manager within their commonly used VLEs, as explained in section 3.5. This issue could be easily solved if the GLUElet Manager had a list with the VLEs for which requests are accepted. That would avoid misuses of the GLUElet Manager, rejecting requests from undesired or malicious clients. Besides, this solution could be useful to control the number of calls made from each VLE. In order to implement this solution, the GLUElet Manager could check the IP of the client that makes the request, comparing this IP with those registered in the list of authorized systems. Nevertheless, IP addresses can be falsified without much difficulty. Better solutions have been implemented in other integration works. For instance, Apache Wookie clients are granted with a unique *API key*, which must be included in every call to an Apache Wookie server. An analogous approach can be thus implemented here for the identification of authorized VLEs. Therefore, VLEs should register themselves in the GLUElet Manager, receiving a *key* that should be included in every call from VLE adapters to the GLUElet Manager.

3. *GLUElet Manager level authorization for the use of tool adapters.* This issue appears when a GLUElet Manager invokes a tool adapter as part of the four use cases explained in section 3.5. Implementing some kind of authorization mechanism would also avoid misuses of tool adapters, serving also to control the number of calls received by each of these adapters. Remarkably, several GLUElet Managers installed in different domains may use the same tool adapter. Therefore, a solution like the one presented for the second issue could be applied here too. In this case, each GLUElet Manager should register itself in tool adapters, which would give them a *key* that should be included in every call from the GLUElet Manager to that tool adapter.

4. *Tool adapter level authorization for the management of external tool instances.* This security issue occurs when tool adapters need to create, delete, and sometimes update, external tool instances. Many external tools require end-users (or other systems) to be authorized in order to create and manage external tool instances. In the context of tool integration, these actions are normally the consequences of educators' behavior within VLEs during the instantiation of learning situations, as previously explained in this chapter. Educators may thus be the ones authorized in external tools, rather than tool adapters, especially if these educators prefer to own the instances they create, so that these instances can occasionally be used (if possible) without VLE or GLUE! mediation. This is an important novelty compared to the second and third security issues, and so, a new issue stems from this one: the *user level authorization for the management of tool instances.* All in all, this is a quite complex problem, since the creation, deletion and update of tools instances need to be delegated to tool adapters on educators' behalf. A more complex solution to tackle this emerging issue is separately proposed and explained in section 3.6.1.

5. *User level authorization for the access to external tool instances.* This security issue arises when end-users retrieve and access to external tool instances. Once instances are retrieved, users directly interact with external tools without the mediation of VLEs or GLUE!. This is because of the fact that the elements of the GLUE! architecture only retrieve URLs representing external tool instances from tool adapters. Once a URL reaches the user's browser, it is this browser that opens a connection to the tool. At this point, participants need to be authorized to use these instances. Typically, end-users register themselves in external tools (for example, by creating a user account) in order to access to their data and resources. To identify authorized users, external tools usually implement some authentication mechanisms. As an example, most tools support a manual authentication employing user credentials. Besides, some external tools support OpenID [Har07a], which is a popular delegated authentication mechanism that enables end-users to authenticate with the credentials of an OpenID identity provider (e.g. Facebook, Google, Yahoo! or MyOpenID[14]). It is noteworthy that, since URLs are used to access to external tool instances, then web browsers may maintain sessions with external tools, and so, end-users would not need to set credentials every time they access to an instance of the same tool in the same session. Significantly, tools may implement different authorization levels, which can be offered to end-users depending on the implementation of tool adapters. An illustrative example is Google Docs, which enables three authorization levels for sharing instances. With the basic mechanism, instances are publicly offered on the Web, and anyone can find and use them. A more elaborated mechanism publicly offers instances too, but only those that have the link (URL) can use them. Finally, there is an option in which the owner of the instances decides who is authorized to use them. All in all, this dissertation relies on the native tool mechanisms for the authentication and authorization of end-users (once they retrieve external tool instances and want to use them embedded with the VLE interface).

6. *Privacy and integrity along the communication channels.* HTTP requests and responses are sent throughout the elements of the architecture using computer networks. Therefore, some policies should be adopted to prevent third-parties from eavesdropping. Classical solutions include the encryption of HTTP requests and responses using asymmetric-key algorithms (public/private key algorithms) [Fer03]. As an example, Basic LTI employs a simple encryption based on a symmetric-key algorithm, namely 2-legged OAuth [IET10], in order to encrypt the `POST` messages sent from *tool consumers* (VLE side) to *tool providers* (tool side). Similarly, two-legged OAuth could be employed here to encrypt the communication between VLE adapters and the GLUElet Manager, and also between the GLUElet Manager and tool adapters.

[14] `http://myopenid.com`. Last visited: June 2012.

3.6.1. User level authorization for the management of external tool instances

The user level authorization for the management of external tool instances is a quite complex security issue that requires a well thought-out solution, and so, this issue, unlike the others, needs a further explanation. This issue refers to educators delegating the creation, deletion and update of tool instances to tool adapters. That entails trusting, at least, in these elements of the architecture, which are the ones that actually execute these actions. Thus, trust risks may appear, since tool adapters are typically offered as distributed systems, being installed and maintained by different providers. There are several delegation mechanism supported by external tools (e.g. the proprietary AuthSub[15] from Google). Fortunately, OAuth [IET10] is gaining momentum as a popular delegation mechanism that enables third-party applications to access data and information in different systems on users' behalf (receiving these applications *access tokens* to be used when they send requests to OAuth compliant systems). OAuth is currently implemented in several important tools like Google Docs or Twitter. OAuth 1.0 is the reference protocol, although there is a new version in draft, OAuth 2.0 [NWG11], and some providers like Google have started to support both versions.

Remarkably, security issues are out of the main scope of this dissertation, although two compromise solutions with different levels of complexity, coupling, and security risks are introduced here. The first one is a weak approach based on **centralized institutional credentials**. These credentials are obtained from external tools by the GLUE! administrator, and allow VLE users to manage tool instances. The GLUE! administrator registers these credentials in the internal tool registry, associating them to specific external tools. From then on, the GLUElet Manager includes these credentials in the Atom feed (in the "*parameters*" field), every time a creation, update or deletion request is sent to tool adapters. Tool adapters use these credentials, as determined by tool contracts, when needed. This first approach has two limitations. First, institutions own external tool instances (i.e. instances are created with institutional credentials), instead of the educators that request their creation. Besides, some tool providers may disagree with this policy, if they consider that credentials must belong to individual end-users, rather than to institutions. Therefore, this approach is intended for small institutions with few educators (and usually with their own GLUE! installation). Besides, this first approach is also recommendable for those educators that do not want to be aware of security issues, insofar as this approach grants SSO once educators are logged in VLEs.

The second is a stronger solution in which educators need to be individually authorized in each external tool (typically educators create an account in external tools, receiving their own personal credentials). This solution relies on the main delegated authorization mechanism at writing time, **OAuth**, so that authorized educators delegate the creation, deletion and update

[15] `http://code.google.com/apis/accounts/docs/AuthSub.html`. Last visited: June 2012.

of tool instances to tool adapters using this mechanism. Besides, and in order to avoid the imposition of being an OAuth server to external tools, this approach also supports the management of tool instances using **native tool credentials** and **OpenID credentials**. In both cases, educators need to share their credentials with tool adapters, delegating the actual creation, deletion or update of instances to them. In order to implement this second approach, a new element is defined as part of the GLUE! core. This element, called GLUE! security component (GSC), helps in the management of these three mechanisms. Besides, the GSC may temporarily store educators' credentials (OpenID or native) or access tokens (obtained with OAuth), to speed up the management of multiple tool instances. The novelties this second approach adds, exemplified in the first use case presented in section 3.5.1 (creation and configuration of external tool instances within VLEs), are the following:

- When implementing tool adapters, developers must indicate in the configuration template one or more mechanisms supported (OAuth, OpenID and/or native tool credentials).
- In the response to the `GET /configuration` (2.4), the GLUElet Manager queries the GSC, which checks whether valid credentials or tokens have been previously stored associated to the educator that wants to create instances (new interaction 2.4.1).
- If the educator has valid credentials or tokens the process goes on as usual. Otherwise, the creation of instances (3.2) is stopped until the educator solves the security dependencies.
- The VLE adapter warns the educator (new interaction 3.1.1), forcing him to click on a new *solve security dependencies* option, which is collected by the VLE adapter. The VLE adapter queries the GSC (3.1.2), which provides the OAuth interface (3.1.3a), so that the educator allows the tool adapter to access the external tool on his behalf (3.1.4a), or a generic authentication interface (3.1.3b) in which the educator sets his OpenID or native credentials (3.1.4b). Both interfaces are exposed as URLs, thus being embedded in the VLE user interface.
- At this point, credentials or tokens may be stored in the GSC at educators' choice. To reduce security leaks tokens expire after a certain time. The same policy can be applied to native or OpenID credentials stored in the GSC.
- Once the dependencies are solved, the process goes on as usual (3.2), although in the `POST` or `DELETE` requests, the GLUElet Manager queries the GSC (new interaction 3.2.1) to obtain the credentials or tokens that may be used.

This second solution requires minor changes in the GLUE! contracts (to include the security mechanisms supported in the configuration templates, to pass the "*callerUser*" in the `GET /configuration` request, and to embed new URLs in VLE adapters). Therefore, this approach

preserves the loosely-coupling in GLUE!, and overcomes the limitations of the first solution, although it does not grant SSO. This solution is intended for bigger institutions with many educators, or where the security in the management of instances is a key issue.

These are two compromise solutions for this security issue. The idea is to let practitioners and institutions choose which of them to employ. They must consider that the first is a weaker solution that hides the security to end-users, at the cost of requiring GLUE! administrators to take care of institutional credentials. The second is a stronger solution that requires end-users to manually solve security dependencies. Developers may incrementally implement their adapters to first meet the weak approach, and then improve their tool adapters to support OAuth, and their VLE adapters to embed the new security interfaces (i.e. the OAuth interface and the generic authentication interface that collects native or OpenID credentials).

Nevertheless, these solutions are not without problems. Practitioners need to trust in some of the GLUE! elements in both cases (the GLUElet Manager, tool adapters, and even the GSC in the second solution). Risks concerning these elements can be reduced by using OAuth, by installing the whole architecture in the institution that is intended to use it, or by trusting only in official GLUE! partners. These partners could be listed in the GLUE! official registry, being certified by external certification authorities [Ala10b]. Despite these risks, it is noteworthy that these are compromise solutions aimed at minimizing the restrictions imposed on VLEs and tools, and the requirements needed for the development of VLE and tool adapters.

3.7. Discussion

The GLUE! architecture aims at meeting the six stakeholders' requirements that were introduced in chapter 2. Now that GLUE! has been presented, a theoretical discussion analyzing which of the decisions taken in the design of GLUE! foster the compliance to each of these requirements can be performed. Remarkably, this is only a theoretical discussion, and evidences backing up the actual compliance to the stakeholders' requirements are provided in chapter 5 of this dissertation.

3.7.1. Compliance to the stakeholders' requirements

The GLUE! architecture is designed to support the life cycle of external tools, including the four typical use cases detailed in section 3.5. The most outstanding features of these use cases are the creation of separated instances for different groups, and the update of the group members allowed to access these instances. As a result of GLUE! supporting this life, educators can easily create and manage tool instances within a single environment, as part of the instantiation of their collaborative learning situations, thus saving a lot of time and effort. As an example, an

educator that does not use GLUE!, but still want their students to use an external tool like Google Documents in a learning activity defined in a VLE like Moodle, must follow several manual steps: go to Google Documents; set valid credentials; create a new instance; configure the instance (e.g. uploading an initial file); modify the access and edition settings; copy the URL of the instance; paste the URL of the instance in Moodle; repeat the last five steps for each group defined in the Moodle activity. Therefore, this educator has to assume an important burden, which increases as the number of students, groups, activities and tools do. Without GLUE!, the educator would probably consider that this burden is not worthwhile, thus simplifying the original learning design to be able to instantiate it with the existing VLE built-in tools. Moreover, this manual management of external tool instances is not always possible because not all the tools are web tools to provide URLs identifying tool instances, as explained in chapter 2. Thus, the overall behavior considered in the design of the GLUE! architecture is intended to meet **REQ1, to enable the instantiation of individual and collaborative activities that require the integration of external tools with an attainable effort for educators**. Besides, by supporting this life cycle, students can find all the external tool instances they need, in order to work in groups within a single and centralized platform, and so, they can focus on achieving the learning objectives, rather than on spending time in searching tools and finding ways to share their contributions with their group partners. Therefore, the behavior considered in the design of GLUE! also fosters the compliance to **REQ2, to enable the enactment of collaborative activities that require the integration of external tools, facilitating the collaboration among participants**.

Both the GLUE! integration contracts for VLEs and tools are designed to impose few restrictions with a widespread adoption to VLE and tool providers. Due to this widespread adoption, VLEs and tools are, in general, likely to meet these restrictions, and so, they could eventually be integrated through the GLUE! architecture. Sections 3.4.2 and 3.4.3 show with outstanding examples in Tables 3.3 and 3.6 that the architecture is designed to foster the compliance to **REQ 3, to support the integration of existing and popular VLEs and tools**. Actually, both mandatory and optional restrictions are met by the seven main VLEs and the seven main tools for education, except for Prezi, which, at the moment, does not meet the optional restriction for tools. Besides, not only these seven tools could eventually be integrated, but also most of those listed in the Top 100 Tools for Learning, as explained in section 3.4.2, thus also promoting the compliance to **REQ 4, to support the integration of many external tools**.

The GLUE! integration contracts for VLEs and tools promote the loosely-coupling through the use of web technologies and popular standards (REST, Atom, XForms and URLs), with the exception of HTML5 which is not a standard yet (although XForms can alternatively be employed). Besides, the GLUE! core acts as an intermediate software layer, partially assuming

the integration functionality and fostering a many-to-many integration of external tools in VLEs. These design decisions aim at facilitating the development of VLE and tool adapters, and thus, they are intended to meet **REQ 5, to demand an attainable development effort for the integration of tools and VLEs**. Remarkably, VLE and tool adapters wrap VLEs and tools by means of the adapter pattern, which benefits the compliance to **REQ 6, to be built over existing VLEs and tools, rather than modifying their implementations**.

A further discussion about the compromise between the development effort and the functionality offered could be argued, taking positions for a higher degree of functionality offered, at the cost of an additional effort. Here, two clarifications can be made. First, GLUE! offers more functionality (in particular the support of the tool life cycle) than other loosely-coupled integration works like Basic LTI. Actually, GLUE! could act as a middleware architecture for loosely-coupled approaches, allowing the integration of Basic LTI compliant tools and W3C widgets in VLEs. Second, GLUE! is designed as a modular architecture. Thus, extra elements could be added to the GLUE! core, imposing new requirements on the adapters and making them thicker. Of course, external developers may decide to implement the current integration contract, or the one that considers the extra functionality. For instance, an extra module could be designed to retrieve and manage outcomes from students' work, but that would probably require tool adapters to expose new REST resources, and VLE adapters to enable educators to collect and visualize those evidences within VLEs.

3.7.2. GLUE! interoperability with other loosely-coupled integration approaches

The GLUE! architecture presents important advantages for the generic integration of external tools in VLEs, compared to other tighter integration approaches in the literature, as previously explained. Besides, GLUE! also presents two important advantages compared to loosely-coupled integration works like Apache Wookie and Basic LTI. These advantages are the imposition of few restrictions with a widespread adoption (as opposed to Apache Wookie), so that a wider range of external tools can be integrated, and the support of the tool life cycle (as opposed to Basic LTI), so that the instantiation and enactment of collaborative learning situations within VLEs is facilitated.

Nevertheless, the GLUE! architecture should not be seen as a competitor of other loosely-coupled integration approaches. Actually, GLUE! may work as a middleware architecture for Apache Wookie and Basic LTI. The reason is that they all impose similar technical requirements; in particular, the definition of REST interfaces. Besides, the functionality offered by both Apache Wookie and Basic LTI can be included in the tool life cycle supported by the GLUE! architecture. More specifically, Apache Wookie supports the creation and configuration

of W3C widget instances, which can later be used and deleted through REST requests to the resources offered by a widget server[16]. Significantly, Apache Wookie also supports the update of users sharing instances. On the other hand, Basic LTI only supports the use of instances [IMS10b] through a `POST` request to a Basic LTI compliant tool.

Figure 3.7 presents how interoperability between GLUE! and other loosely-coupled integration approaches could be achieved. W3C widgets could be integrated in VLEs by developing an Apache Wookie adapter (plug-in in the Apache Wookie terminology [Wil08]), which connects the GLUElet Manager and a widget server, in which W3C widgets are deployed. According to Wilson's design [Wil08], the widget server provides a homogeneous interface that enables the creation and configuration of W3C widgets, whose data and user preferences (configurations) are stored in this server. New widgets might be deployed in the widget server, and they could be integrated in VLEs through GLUE! using the same Apache Wookie adapter. Similarly, Basic LTI compliant tools could be integrated in VLEs by developing a Basic LTI adapter (consumer in the Basic LTI terminology [IMS10b]). This adapter would connect the GLUElet Manager and Basic LTI providers, which wrap external tools, so that these tools may meet the homogeneous Basic LTI integration contract; new Basic LTI compliant tools could also make use of the same generic Basic LTI adapter. The concept of GLUE! as a middleware architecture might be extended to other similar loosely-coupled integration approaches that follow similar technical requirements, and whose functionality fits within the tool life cycle supported by GLUE!.

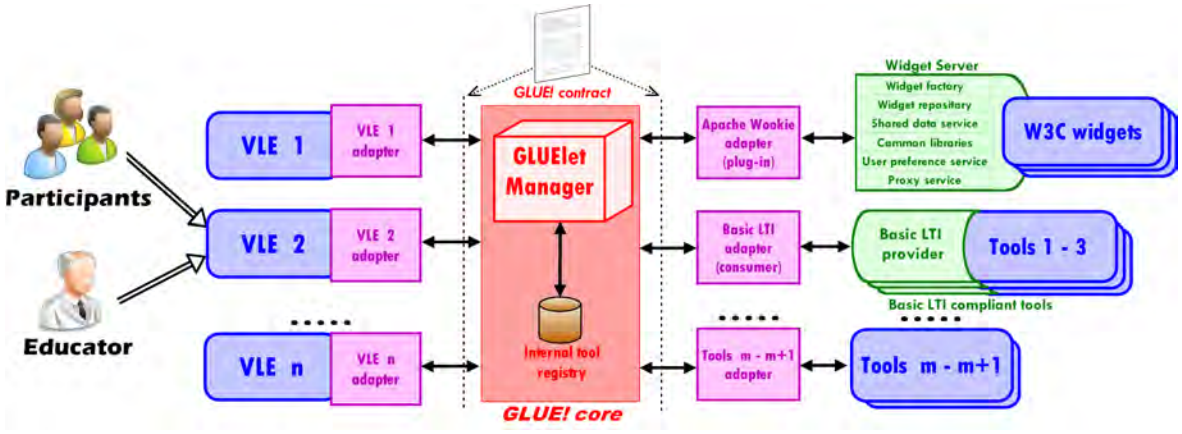

Figure 3.7: GLUE! interoperability with other loosely-coupled integration approaches: Apache Wookie and Basic LTI.

The use of GLUE! as a middleware architecture presents several advantages compared to the direct integration of W3C widgets and Basic LTI compliant tools. First, the development effort is reduced, since the code required for the development of the Apache Wookie and Basic LTI adapters for GLUE! is typically lower (compared, for instance, to the effort required to develop an

[16] `https://cwiki.apache.org/WOOKIE/wookie-rest-api.html`. Last visited: June 2012.

Apache Wookie plug-in for Moodle or a Basic LTI consumer for Blackboard), since the graphical integration with VLEs is assumed by VLE adapters, and the management of instances is partially covered by the GLUElet Manager. Besides, once these two adapters are developed, W3C widgets and Basic LTI compliant tools become available in multiple VLEs, taking advantage of the many-to-many integration. Another advantage is that the information of the available external tools is centralized in the internal tool registry, and so, VLE administrators do not need to add and configure every Basic LTI compliant tool in each VLE (as it currently occurs), thus reducing their administrative burden. Finally, the interoperability of GLUE! with other loosely-coupled approaches increases the range of available tools for educators and students in the instantiation and enactment of collaborative learning situations.

3.7.3. GLUE! for the integration of external tools in other contexts

The decisions taken in the design of the GLUE! architecture can also be applied to achieve interoperability among software systems in other contexts where collaboration and integration are key issues. These systems may belong to the educational domain, but also to others where an integration solution should be built upon similar requirements to those presented in section 3.3.

For instance, wikis are software platforms used in education to support collaboration and groupwork, although they can also be employed in other domains like CSCW [Ell91], in order to achieve a common understanding within a group or community. MediaWiki, the software application used in most wikis, was designed to support collaboration and the integration of external applications, and so, it could benefit from the design decisions behind the GLUE! architecture. Moreover, MediaWiki meets the mandatory restrictions of GLUE!, as it can be seen in Table 3.10. Therefore, a MediaWiki VLE adapter could be developed to integrate external tools in MediaWiki through the GLUE! architecture. As an example, the integration of Text2MindMap in a MediaWiki page to promote the collaborative organization of ideas over a certain topic would be feasible through the GLUE! architecture. Nevertheless, MediaWiki lacks support to the creation and management of groups, thus limiting the functionality that might be offered, as it will be later discussed in the implementation chapter.

Other examples are social networking sites, which are mainly used for social interactions. Nevertheless, they have also been used to support collaboration and communication in the educational domain [Loc08]. Facebook, the most widespread social networking site, was designed to be extended through the development of third-party *plugins*, as many other of these sites do, and so, it could benefit from the design decisions behind the GLUE! architecture. Furthermore, Facebook meets the mandatory restrictions of GLUE!, as it can be seen in Table 3.10, and thus, a Facebook VLE adapter to integrate external tools in Facebook through GLUE! could be de-

Table 3.10: Restrictions imposed on other platforms used for learning and degree of adoption. Mandatory restrictions are marked in bold, while optional restrictions are marked in italics.

Restrictions	***MediaWiki***	***Facebook***	***SLE***
Render web contents	✔	✔	✔
Extension interface	✔	✔	✔
Concept of tool	✔	✔	✔
Concept of group	✘	✔	✔
Concept of role	✔	✘	✘

veloped. As an example, the use of Doodle as a poll tool within Facebook in order to schedule a meeting with old friends would be feasible through the GLUE! architecture. Nonetheless, Facebook lacks support to the definition of roles, and so, the integration of tools would occur in informal and non-hierarchical contexts, where the life cycle of collaborative learning situations is blurred, and users arrange the integrated tools by themselves.

Similarly to social networking sites, PLEs are designed to support informal education where learners aggregate and configure tools and resources by themselves. PLEs are intended to support collaboration and communication too, and so, they could benefit from the design decisions in the GLUE! architecture. Actually, the SLE [Whi11], which is one of the most remarkable examples of PLE, meets the mandatory restrictions of GLUE!, as presented in Table 3.10. Therefore, a SLE VLE adapter could also be developed to promote the integration of external tools in this PLE. As an example, Google Presentations could be integrated in SLE through GLUE! in order to promote the realization of a collaborative presentation among a set of students freely arranged according to their common interests.

Regarding the applicability of the ideas behind the GLUE! architecture in domains other than education, they can also be useful in the Business Process Modeling domain [Geo95], where many research efforts have been devoted to the proposal of workflow engines, such us Bonita[17], Together Workflow Server[18], or Joget[19]. These engines can interact with processes and tools [Pal07], which are currently integrated following a rather *ad hoc* approach, and so, they could benefit from an integration architecture similar to that proposed in this dissertation.

3.8. Conclusions

The GLUE! architecture is designed to enable the integration of multiple existing external tools in multiple existing VLEs, imposing few restrictions that most VLE and tool providers

[17] `http://bonitasoft.com`. Last visited: June 2012.
[18] `http://www.together.at/prod/workflow/tws`. Last visited: June 2012.
[19] `http://joget.org`. Last visited: June 2012.

currently meet, reducing the development effort required to integrate new VLEs and tools, and supporting the life cycle of external tools within VLEs. The decisions made during the design of GLUE! consider the issues and alternatives analyzed in the integration problem, as well as the limitations found in previous related works.

The support that GLUE! provides for the management of the life cycle of external tools is intended to allow the efficient and successful instantiation (REQ1) and enactment (REQ2) of collaborative learning activities in which such tools are employed. The imposition of few and popular restrictions to VLE and tool providers (three mandatory restrictions on VLEs and one mandatory restriction on tools, plus two optional restrictions on VLEs and one optional restriction on tools) try to facilitate the integration of many (REQ4) existing and popular external tools in the main VLEs (REQ3). Besides, the definition of loosely-coupled integration contracts, which combine REST interfaces, Atom data format, XForm or HTML5 configuration templates, and URL representations, the proposal of an intermediate software layer that partially undertakes the integration functionality, and the fostering of a many-to-many integration are oriented to reduce the development effort (REQ5). Finally, by means of the adapter pattern existing VLEs and tools can be wrapped without modifying their code (REQ6). Significantly, security issues, which entail an additional problem in generic integration approaches in this context due to the heterogeneity of VLEs and tool, have been analyzed proposing a compromise solution based on widely accepted standards too. Figure 3.8 summarizes the requirements, the design decisions and the consequence these decisions have in the GLUE! architecture.

The GLUE! architecture can clearly be seen as an integration alternative to tighter integration works that require a much higher development effort. Nevertheless, it can also complement loosely-coupled integration works, like Apache Wookie or Basic LTI. Actually, the GLUE! architecture could complement other loosely-coupled integration works that employ similar technologies, and whose functionality offered fits in the life cycle of external tools. Though GLUE! is an integration solution for the context of software tools and VLEs, it could also be applied to other platforms used for collaborative learning like wikis, social networking sites or PLEs, and in other domains apart from education, such as Business Process Modeling or CSCW.

It is noteworthy that the global objective of this dissertation is the design, development and evaluation of a middleware architecture that enables the integration of multiple existing external tools in multiple existing VLEs, overcoming the limitations of previous related works. This chapter has presented the first step, which is the design of the architecture. Next chapters deal with the development of a reference implementation and with the evaluation of the architecture using this reference implementation.

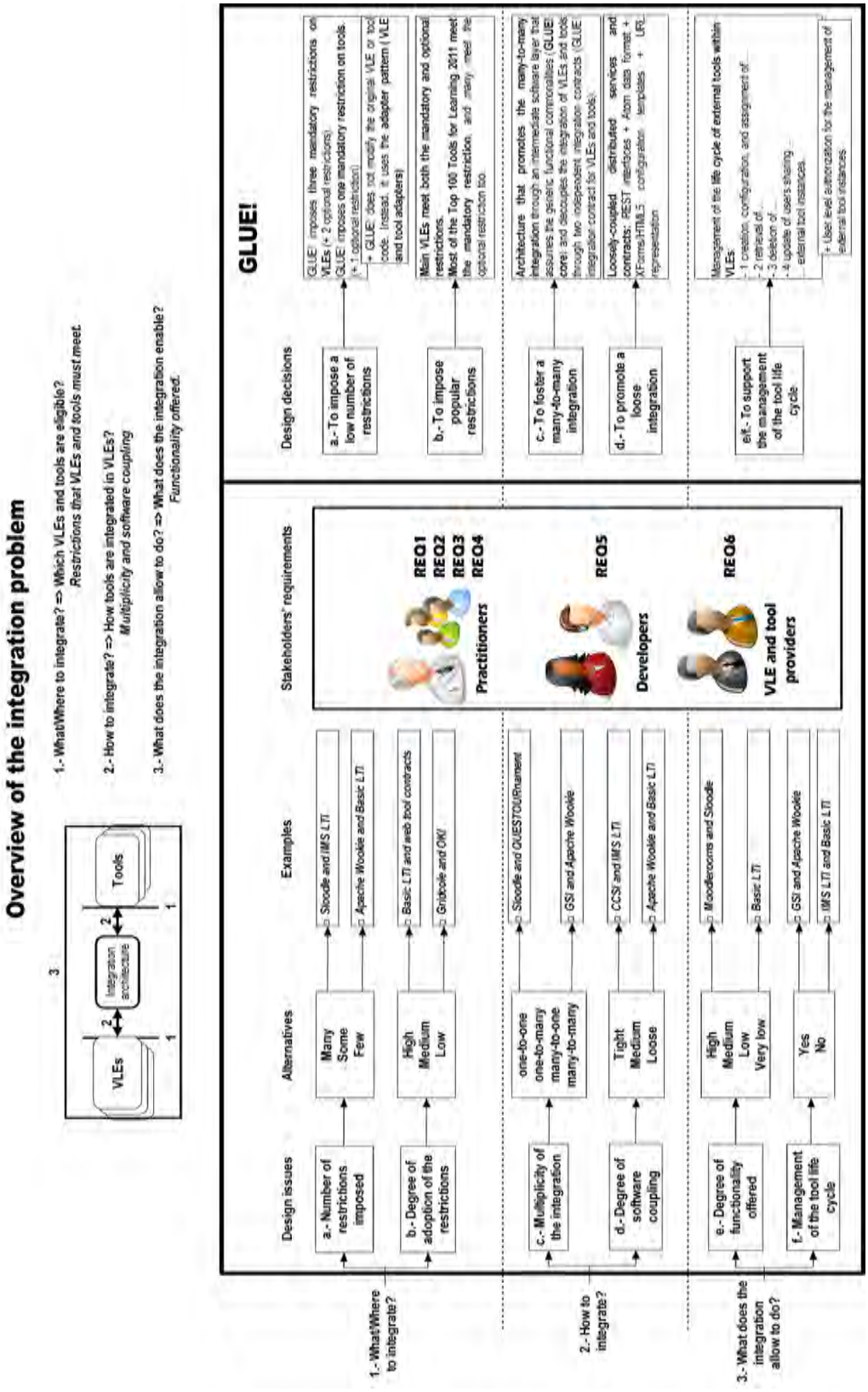

Figure 3.8: Overview of the proposed architecture within the integration problem.

Chapter 4

GLUE!-RI: Reference implementation of the architecture

This chapter presents the reference implementation of the GLUE! architecture, which includes a reference implementation of the GLUE! core and several VLE and tool adapters. This reference implementation shows that the GLUE! architecture can actually be implemented. Besides, it is useful to formally evaluate whether GLUE! meets the stakeholders' requirements, and overcomes the limitations of the related works. Furthermore, it starts up a collection of VLE and tool adapters that may grow with external contributions. In order to properly organize this chapter, section 4.2 introduces the methodology employed for the development of the reference implementation. Then, section 4.3 overviews the elements that compose the reference implementation of the GLUE! core and the available adapters. After that, section 4.4 provides instructions for third-party developers that want to integrate new tools or VLEs through the GLUE! architecture. Additionally, section 4.5 explains the steps for installing and configuring this reference implementation, while section 4.6 illustrates its usage from end-users' perspective. Finally, the chapter concludes with section 4.7.

The reference implementation of the GLUE! architecture is first mentioned in [Ala12a]. The code of the GLUE! core and the available adapters can be found at `http://gsic.uva.es/glue`, being this set of adapters continuously growing. Besides, the particular challenges involved in the development of a LAMS adapter (considering the LAMS distinctive features) have been published in the specific international conference of this VLE [Ala11b], while [Ala12c] illustrates the instantiation and enactment of an authentic collaborative learning situation in both Moodle and LAMS.

4.1. Introduction

Once the GLUE! architecture has been fully specified, any developer could proceed to implement it. Nevertheless, to facilitate this work, and in order to assess GLUE! in real practice, a reference implementation of GLUE!, called GLUE!-RI (GLUE! Reference Implementation), has been developed in the context of this doctoral research; GLUE!-RI is the generic term for a software distribution that includes a reference implementation of the GLUE! core and a few

examples of VLE and tool adapters; all these elements are further detailed in this chapter. A reference implementation is by definition a complete and usable implementation of a specification or architecture, "which is warranted to give the authors' intended answers in a moderately-sized problem" [Rip02]. In the case of specifications, a reference implementation should be a precise and unambiguous interpretation of the specification [Pos00]. Significantly, reference implementations of specifications are normally thought for developers, and not so much for end-users. As an example, IMS provides a reference implementation for Basic LTI in PHP, Python and Java[1], aimed at supporting developers working in Basic LTI, and also at promoting the integration of this specification in other applications. In the case of software architectures, a reference implementation should be a fully functional system intended for developers, but also for end-users, so that they can use the proposed system in authentic scenarios. As an example, Apache Wookie (Incubating)[2] is the open source reference implementation of the architecture presented in [Wil08]. In both cases, reference implementations could be useful as models for further comparisons and evaluations with other implementations of the same specification or architecture. In summary, reference implementations offer some straightforward code, which should be available to everyone intended to study, develop, use or evaluate a specification or architecture.

The development of the reference implementation of the GLUE! architecture at this point is highly advisable for several reasons. First, it shows that GLUE! can actually be implemented. Second, it can be tested and evaluated to study whether GLUE! overcomes the limitations of previous works. Third, it can be used by real practitioners to support collaborative learning situations within VLEs, which is the final purpose intended for this architecture. Finally, it can be used to start up a collection of VLE and tool adapters that grows through the contributions of external developers, who may implement new adapters. This is a key issue, since generic integration works normally rely on the creation of communities of developers, who share their implementations, thus working to increase the number of integrated systems. In this case, these contributions may help practitioners to choose from a wider variety of tools during the instantiation of their learning designs, no matter the VLEs they are used to.

Being GLUE!-RI just the reference implementation, other implementations are feasible too. Actually, each of the tiers of GLUE!-RI (GLUE! core, VLE adapters, tool adapters) could be improved by supporting new use cases built on the top of the existing ones, and new tools and VLEs could be integrated by developing new adapters; as long as these elements meet the GLUE! integration contracts (and the VLE or tool contracts), they would interoperate with the other elements developed as part of GLUE!-RI. The code of GLUE!-RI is available at `http://gsic.uva.es/glue` for its improvement, or to be reused when programming new adapters.

[1] `http://code.google.com/p/ims-dev`. Last visited: June 2012.
[2] `http://incubator.apache.org/wookie`. Last visited: June 2012.

The remainder of this chapter presents the reference implementation of the GLUE! architecture, focusing on each of the elements that were developed so far. Besides, this implementation is analyzed from the perspectives of three different kinds of users: developers that may want to contribute to the current implementation; administrators that may want to install and configure GLUE!-RI; and practitioners that may want to use GLUE!-RI to integrate external tools in their collaborative learning situations.

4.2. Methodology

The overall methodology followed to define the requirements, design, implement, test and deploy the GLUE! architecture is the UP [Lar02], as mentioned in the previous chapter. However, the activities related to the development of GLUE!-RI only include: implementation, testing and deployment. The components of GLUE!-RI are separately developed, as recommended in the UP, being later tested and deployed altogether in order to check whether the different elements meet the initial requirements and the expected behavior. Besides, since the UP is the methodology employed, then GLUE!-RI is incrementally and iteratively developed, adding new use cases and functionality in each iteration, as it is detailed in section 4.3. Significantly, the number of VLEs and tools that are integrated through GLUE! is also incrementally augmented by developing new adapters.

Besides, the **Scrum framework** [Kni06] for project management and agile software development in its variant for small teams [Ris00] is employed in order to facilitate the activities related to the implementation, testing and deployment of GLUE!-RI. Scrum is a framework intended for the management of complex software development projects, as it is the case of GLUE!-RI, which involves multiple technologies, contracts and systems. According to Scrum, complex projects should be divided in short and homogeneous periods of time, called *sprints*, which may last from one week to a few months. Each sprint contains a list of tasks to be tackled. These tasks are prioritized in an earlier meeting, being formally arranged in *backlogs* (one for each sprint, and one for the whole project). Each sprint ends with another meeting in which the degree of accomplishment of each of these tasks is presented, and the backlogs are updated. In the particular context of GLUE!-RI, three people constitute the development team, including the author of this dissertation. Furthermore, in this case, the duration established for the sprints is one week, and different project backlogs are annotated: one for the GLUElet Manager, one for each VLE adapter, and one for the whole set of tool adapters. The use of a popular framework like Scrum promotes a better identification of priorities depending on the expected research interests and evaluation outcomes, reduces overtime, and facilitates the coordination among researchers and the members of the development team (and any other external developer that may later join).

4.3. Reference implementation

GLUE!-RI includes a reference implementation of the GLUE! core and several examples of VLE adapters and tool adapters. The technologies employed for the development of these elements are first introduced in this section. Then, a global overview of GLUE!-RI is offered, detailing later each of the available elements.

4.3.1. Technologies

The GLUElet Manager and the tool adapters are designed as REST services, exposing a set of resources through uniform interfaces, as studied in the previous chapter. In order to develop these REST services, a RESTful web framework for Java, called Restlet[3], is employed. Restlet was selected because it is open source, and one of the main frameworks used nowadays for the development of RESTful web services [Lou12]. Besides, Restlet includes a package to support the Atom syndication format[4], thus facilitating, not only the management of resources and URIs, but also the communication and transmission of data among the elements of the GLUE! architecture. Furthermore, due to the use of the Java programming language, configuration templates can be easily processed using the standard Java package for parsing XML documents[5]. The use of Restlet facilitates the development of GLUE!-RI, but also the development of new adapters, since much of the code is included in this framework. Moreover, the use of Restlet also promotes the robustness and standardization of the reference implementation because this is a leading framework. Nonetheless, it is convenient to remark that these REST services could be developed using alternative frameworks and languages, as long as they meet the GLUE! integration contracts.

VLE adapters, however, are not REST services, although they are expected to invoke the RESTful interface of the GLUElet Manager. Besides, VLE adapters are normally developed in the same language that the VLE they wrap, since these adapter normally run in the VLE server, extending the VLE functionality, and possibly modifying its graphical user interface. Therefore, only those VLE adapters wrapping VLEs developed in Java, like LAMS or Sakai, can use the Restlet framework. Nevertheless, other frameworks, like the Recess framework[6], which is an outstanding RESTful PHP framework, could be employed to facilitate the development of some other VLE adapters, like those for Moodle or Claroline.

[3] `http://restlet.org`. Last visited: June 2012.

[4] `http://restlet.org/documentation/snapshot/jse/ext/org/restlet/ext/atom/package-summary.html`. Last visited: June 2012.

[5] `http://docs.oracle.com/javase/1.4.2/docs/api/javax/xml/parsers/package-summary.html`. Last visited: June 2012.

[6] `http://recessframework.org`. Last visited: June 2012.

4.3.2. Overview

The reference implementation of the GLUE! core and the available set of VLE and tool adapters at the time of writing this document are shown in Figure 4.1. As it can be seen, three VLE adapters and nine tool adapters have been implemented. The three VLE adapters enable the integration of external tools in Moodle, LAMS and MediaWiki. The nine tool adapters enable the integration in VLEs of Google Docs (Documents, Spreadsheets and Presentations), MediaWiki, Dabbleboard, W3C widgets deployed in Apache Wookie servers, Doodle, Facebook Live Stream[7] (a forum for Facebook users), Kaltura, Noteflight[8] and any URL representing a web content.

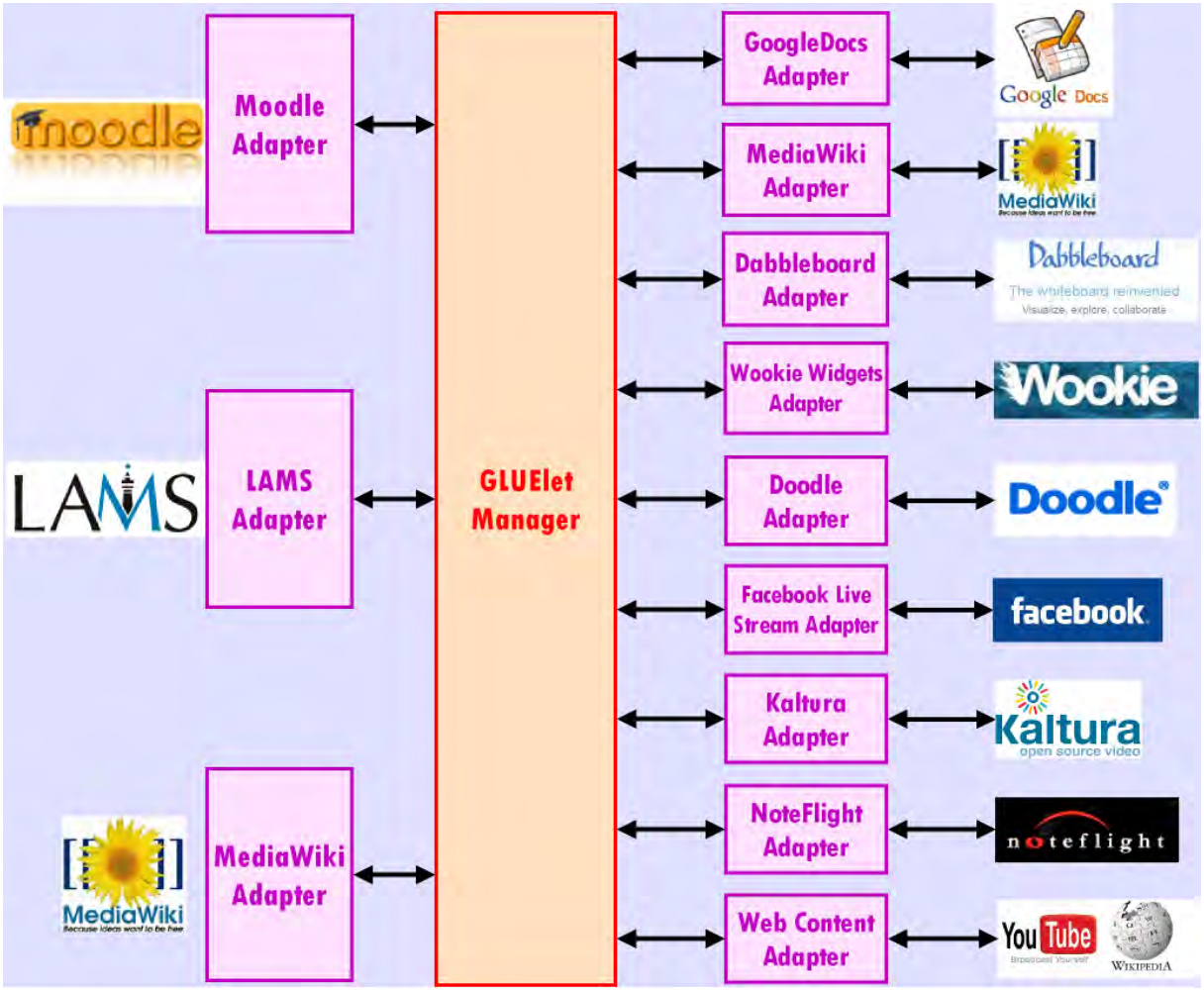

Figure 4.1: Overview of GLUE!-RI including: the GLUElet Manager, three VLE adapters (Moodle, LAMS, MediaWiki) and nine tool adapters (Google Docs, MediaWiki, Dabbleboard, Apache Wookie, Doodle, Facebook Live Stream, Kaltura, Noteflight, web content).

[7] `http://developers.facebook.com/docs/reference/plugins/live-stream`. Last visited: June 2012.
[8] `http://noteflight.com`. Last visited: June 2012.

GLUE!-RI is incrementally and iteratively developed. Actually, Table 4.1 shows the five iterations that have been carried out on GLUE!-RI. The first iteration included the development of the GLUElet Manager, the Moodle adapter and three tool adapters (Google Docs, Dabbleboard and web content), and was able to support the basic tool life cycle that was originally presented in section 2.4.2, using the first solution for the user level authorization in the management of external tool instances (i.e. weak solution based on institutional credentials). The second iteration added one new VLE adapter (MediaWiki) and two new tool adapters (MediaWiki and Apache Wookie), and also implemented the use case concerning the update of users, which stemmed from the results of a first evaluation experiment with real practitioners. The third iteration added a new tool adapter (Doodle) and implemented a new use case to support the reuse of instances in different activities of a collaborative learning situation instantiated in Moodle or MediaWiki. The lack of functionality to reuse instances was also detected in the first evaluation experiment with real practitioners. This new use case relies on VLE adapters and the behavior supported by the GLUElet Manager and tool adapters, and so, it does not modify the design of the architecture, nor the integration contracts, as it is further explained in section 4.3.4. The fourth iteration included the development of a new VLE adapter (LAMS) and three new tool adapters (Facebook Live Stream, Kaltura and Noteflight). Finally, the fifth iteration implemented the second solution for the user level authorization in the management of external tool instances (i.e. strong solution supporting OAuth, OpenID, and native credentials). With the fifth iteration GLUE!-RI achieves the status it was intended for. Nevertheless, new VLE and tool adapters may be developed in subsequent iterations, as well as new use cases or other proposals regarding, for instance, security issues. Actually, the solutions presented in chapter 3 for the VLE level authorization in the use of the GLUElet Manager, for the GLUElet Manager level authorization in the use of tool adapters, and for the privacy and integrity along the communication channels, are pending for implementation, but with a lower priority.

Table 4.1: Iterations in the development of GLUE!-RI.

Number of iteration	New VLE adapters	New tool adapters	New use cases implemented	User level authorization for the management of instances	End date
1	1 (Moodle)	3 (Google Docs, Dabbleboard, web content)	3 (creation and configuration, use, and deletion of external tool instances)	Weak solution	October 2010
2	1 (MediaWiki)	2 (MediaWiki, Apache Wookie)	1 (update of users)	Weak solution	April 2011
3	-	1 (Doodle)	1 (reuse of instances)	Weak solution	July 2011
4	1 (LAMS)	3 (Facebook Live Stream, Kaltura, Noteflight)	-	Weak solution	November 2011
5	-	-	-	Strong solution	June 2012

The duration of each of the five iterations is quite variable, as it can be seen in the end date associated to each iteration in Table 4.1. The product obtained after the end of each iteration is then tested and deployed by the development team, and later used by real practitioners in a different experiment, as detailed in chapter 5. Once each product is tested, deployed and used, the elements belonging to that product are improved to address the problems identified. Some of these problems may require further changes in the design of the architecture, as it happened with the update of users, thus iterating over the phases of the overall research methodology and the UP.

GLUE!-RI is licensed under the GNU General Public License (GPL)[9] for non-commercial uses. Therefore, the reference implementations of the GLUE! core and the available adapters can be redistributed and modified under the terms established in the GPL license. Those interested in employing some of these elements for commercial purposes should contact the GSIC-EMIC research group, which is the proprietary of GLUE!-RI. The information regarding the installation, configuration and usage of GLUE!-RI is detailed in sections 4.5 and 4.6.

4.3.3. GLUE! core

The reference implementation of the GLUE! core includes one single processing element in the business logic layer, the **GLUElet Manager**), and two databases in the data layer. The first database is the **internal tool registry**, which was considered in the design of the architecture. This database stores persistent information about the available external tools. The second database is the **gluelets repository**, which is implemented in this reference implementation to locally support the storage and management of information about the external tool instances that are created. Significantly, no presentation tier is available for practitioners, since they make use of the GLUE! core from their commonly used VLEs. However, a presentation tier may be considered in the future to help GLUE! administrators in the process of creating and updating the information in the internal tool registry (registries can currently be queried or updated by GLUE! administrators using existing database management applications).

The *GLUElet Manager* is implemented as a REST service, employing the Restlet framework for Java. The GLUElet Manager exposes the resources detailed in section 3.4.4, and supports the four use cases detailed in section 3.5. Besides, it uses the JDBC (Java Database Connectivity) API[10] for accessing the two aforementioned databases.

The *internal tool registry* is a MySQL database that collects the information regarding the available set of external tools. This database must be populated during the installation of GLUE!, and later updated to include more external tools, for example if new tool adapters are

[9] `http://www.gnu.org/licenses/gpl.html`. Last visited: June 2012.
[10] `http://docs.oracle.com/javase/tutorial/jdbc`. Last visited: June 2012.

developed. Several SQL scripts are available to facilitate the initial population of the internal tool registry. The fields in this registry are arranged as explained in Table 4.2.

Table 4.2: Fields in the internal tool registry.

Name	Purpose
Tool identifier	Integer that uniquely identifies the external tool, and also the REST resource containing the information about this tool. The GLUElet Manager exposes this REST resource under the form of `/tools/{tool_identifier}`. The tool identifier is for the internal use of the architecture only.
Tool name	Official name of the tool (e.g. Doodle) to be displayed to educators and students.
Tool type	Description of the tool using educational abstractions, modeled according to the Ontoolcole ontology [Veg08]. Examples of these descriptions are: "communication tool" and "synchronous text editor". Educators could perform semantic searches of educational tools using these descriptions. These searches turn out especially useful as the number of available external tools in the internal tool registry increases. The semantic search of tools is not available in the current implementation yet, although it is listed for future work.
Tool provider	URL of the company or institution providing the external tool. Some tools are offered by one single vendor (e.g. Facebook or Google Docs), while others can be installed and offered from different domains (e.g. MediaWiki or the widget server). This information is intended for educators when adding external tools, so that they can choose their preferred tool provider.
Tool adapter	URL of the adapter that wraps the external tool. The GLUElet Manager uses this information to address its requests to the correct tool adapter. This information is never displayed to end-users.
Tool parameters	Extra information required to manage external tool instances. These parameters may contain, for instance, the addressing of the tool interface, or credentials for the authentication with external tools in the weak security mechanism described in the previous chapter. This information is stored in the internal tool registry, rather than in tool adapters, so that a minor change (e.g. the IP of the server that provides the tool) does not require a modification in the tool adapter, but an update in this centralized registry. These parameters are for internal use of tool adapters, and so, tool adapters need to know how to manage them.

The *gluelets repository* is the database that collects the information of the external tool instances that are created. This database is initially empty, and it is automatically populated by the GLUElet Manager as instances are created. Each entry in this repository is called *gluelet*, and typically represents one external tool instance in the GLUElet Manager. The fields in this repository are described in Table 4.3.

Table 4.3: Fields in the *gluelets* repository.

Name	Purpose
Gluelet identifier	Integer that uniquely identifies the REST resource containing the information of the external tool instance in the GLUElet Manager. The GLUElet Manager offers this REST resource under the form of `/instance/{instance_identifier}`. This information is for the internal use of the architecture only.
Instance URI	URI that identifies the external tool instance in a REST resource exposed by a tool adapter. The GLUElet Manager uses this information to retrieve, update, or delete the tool instance.
Tool identifier	This is the identifier of the tool that created the instance. This information may be used by the GLUElet Manager to extract the corresponding tool parameters from the internal tool registry, in case they are needed for the update or deletion of the tool instance.

This is the reference implementation of the GLUE! core, although other implementations for the GLUElet Manager and the databases could be developed. Besides, this reference implementation could be improved to add some extra features. For instance, several GLUElet Managers hosted by different institutions could be federated, sharing one single internal tool registry (although each GLUElet Manager may have its own *gluelets* repository). This way, the information regarding external tools would be centralized, and it could be more easily updated by one single administrator.

4.3.4. VLE Adapters

Three VLE adapters for Moodle, LAMS and MediaWiki are available in GLUE!-RI as of this writing. These adapters are first called by the VLE when the user wants to create a new *gluelet* (how this is commanded by the user is VLE-dependent). After that, VLE adapters must interact with the user (to show configuration information or to report the result of the creation), with the VLE (to query the permissions of the user, to retrieve the list of users that must share this instance, or to other VLE-specific needs to be checked upon the creation of an activity or a tool instance) and with the GLUElet Manager. Besides, VLE adapters must store information that map VLE-specific identifiers, such as activity, group, course or tool identifiers (or a combination of some of them) to GLUE! identifiers, namely *gluelet* identifiers (see Table 4.3). Other information concerning the current status of users allowed to access an instance may be persisted too, in order to facilitate their modification and update. All this results in four commonalities of the otherwise different adapters.

- VLE adapters must provide a graphical user interface that is integrated with that of the VLE, preferably keeping the same look-and-feel.
- VLE adapters must meet the integration contract imposed by the VLE they wrap.
- VLE adapters must meet the GLUE! integration contract for VLEs.
- VLE adapters must persist, in the VLE database or in a separate one, the aforementioned information to connect VLE concepts to those of GLUE!.

Significantly, new specific use cases can be built upon these four commonalities without modifying the design of the GLUE! architecture, or the GLUE! integration contracts. As an example, the LAMS adapter supports the export of learning designs that include built-in and external tools in the authoring environment, as later detailed. The export of learning designs is a LAMS distinctive feature, aimed at promoting the sharing of these designs among practitioners, and so, it was considered when designing and developing the LAMS adapter. Significantly, no

extra code (apart from the one required by the LAMS contract) was specifically programmed in the LAMS adapter to support this use case.

Nevertheless, in other situations, some code could be added to VLE adapters to support additional use cases that may be of interest for practitioners. An example is the *reuse of instances* (also reload of instances) in different activities within the same course or lesson, which is currently included in the VLE adapters for Moodle and MediaWiki. This functionality allows educators to indicate that the same tool instance used in an earlier activity of the learning design is intended to be used again in a later activity. As instances involve a state of a resource in the tool (e.g. the state of a document), this can also be seen as stating that the output of one activity becomes the input of a later one (though only if both employ the same external tool).

This use case is useful to instantiate those learning designs where students need to reuse previous results to perform a certain activity. For example, in a peer review, the documents generated by a group of students must be reviewed by another group in a subsequent activity. Many works have researched the so-called data flow problem [Pal08], which studies the flow of artifacts among learning activities. Nonetheless, the solutions to this problem normally entail the addition of much extra *ad hoc* code to process the output and input artifacts. On the contrary, a simpler and more generic approach can be implemented in VLE adapters to achieve instance reuse in VLEs: each time the user requires to create a new activity or tool backed up by a *gluelet*, two options are available: the creation of a new instance or the reuse of an existing one. The use case for creating an instance was detailed in section 3.5.1. If the educator prefers to reuse an instance, the VLE adapter shows a list of the existing ones for this tool and VLE, and the educator chooses the one he finds appropriate. Then, the VLE adapter stores in its repository a mapping between a *new* VLE activity or tool and an *already existing gluelet identifier*. The GLUElet Manager is not invoked in this use case, since no instance is actually created or modified. Thus, this solution preserves the loosely-coupling of the GLUE! architecture.

Interestingly, sometimes the reuse of instances can be further exploited together with the ability of GLUE! to update the users allowed to access an instance at a given time (e.g. to let the members of a group consume a document generated by another group in a previous activity). The reuse of instances is especially indicated in those designs that follow a logical structure or sequence, and where users do not need to make changes in the activities that are finished.

The VLE adapters for Moodle, LAMS and MediaWiki are detailed next. As usual, other implementations for these adapters, which may include extra use cases, are feasible, as long as they meet the GLUE! integration contract for VLEs and the contracts of the VLEs they wrap. Besides, improvements on these adapters, and updates due to changes in the version of VLEs, can also be made. In particular, and besides the specific issues discussed for each of the adapter, all of them process XForms configuration templates (mostly because of the limitations on their rendering by current web browsers), but they have not been evaluated with HTML5 templates.

Moodle Adapter

The Moodle adapter is available for the version 1.9 of Moodle, although a new implementation is planned for the version 2.2 onwards. The code of this adapter is in PHP, and the cURL library for PHP[11] is employed to make the HTTP requests to the REST resources exposed by the GLUElet Manager.

Moodle manages built-in tools in the form of *Moodle activities*, as explained in section 2.3.1. Therefore, the VLE adapter associates the concept of Moodle activity to that of a *gluelet*. The Moodle adapter makes use of the Moodle internal MySQL database, adding new tables to store specific information about the *gluelets* that are created. These tables are similar to those in other modules that extend the functionality and built-in tools in Moodle, and contain the fields detailed in Table 4.4.

Table 4.4: Fields in the tables added by the Moodle adapter to the Moodle database.

Name	Purpose
Local identifier	Integer that uniquely identifies each combination of course, activity, tool, group and *gluelet* (*gluelet identifier*). This identifier is generated and managed by the Moodle adapter.
Course identifier	Integer that uniquely identifies the course in Moodle. The course identifier is automatically generated and managed by Moodle.
Activity identifier	Integer that uniquely identifies the activity in Moodle. The activity identifier is automatically generated and managed by Moodle, and it is independent of the course identifier, even though Moodle activities are normally arranged in courses.
Name of the activity	Name of the Moodle activity. This name is set by the educator when creating the Moodle activity, and it is later shown to end-users when carrying out that activity.
Description of the activity	Description of the Moodle activity. This description is also set by the educator, and it is later shown to end-users when carrying out that activity too.
Name of the tool	Name of the external tool. This information is retrieved from the internal tool registry (querying the GLUElet Manager), and corresponds to the external tool selected by the educator. The name of the tool is also shown to end-users when enacting the activity.
Tool identifier	Identifier of the tool in the internal tool registry for which that *gluelet* is created (e.g. `/tools/{tool_identifier}`). This information is retrieved from the internal tool registry, and corresponds to the external tool selected by the educator. The tool identifier may be internally used by the Moodle adapter (e.g. to reuse instances of the same tool in different Moodle activities).
Group identifier	Integer that uniquely identifies the Moodle group that is expected to share that *gluelet*. The group identifier is automatically generated and managed by Moodle. In those cases where educators do not define groups in an activity, the value of this integer is *-1*, and the *gluelet* is shared among all the participants belonging to the Moodle course.
Gluelet identifier	Identifier of the *gluelet* in the *gluelets repository* (e.g. `/instance/{instance_identifier}`). This information is used by the Moodle adapter to retrieve, delete or update this instance.

The Moodle adapter implements the four use cases supported by the GLUE! architecture, plus the reuse of instances in different activities of the same course. For the latter, the *gluelet identifier* reused is copied to a new entry in the aforementioned table, and a call to the GLUElet

[11] `http://php.net/manual/en/book.curl.php`. Last visited: June 2012.

Manager, in order to update the users sharing that instance, is sent. Therefore, with this adapter, educators can create and manage external tool instances within Moodle during the instantiation of their learning designs, while students can later use these instances to enact the individual and collaborative learning activities. It is noteworthy that educators can also monitor the students' work at any moment during the enactment of the collaborative learning situation, by visualizing the available external tool instances for each group within the Moodle interface. Furthermore, educators can annotate the content of these instance, giving, in most cases, a synchronous feedback to the students. The usage of the Moodle adapter for educators and students, as well as its installation and configuration processes are further detailed in sections 4.5 and 4.6. Among the limitations found in the Moodle adapter, it must be highlighted that the backup option, which enables the creation of backups of Moodle courses, is not implemented in the current version of this adapter. This is an important Moodle feature in order to recover from failures, and so, it is listed for further versions of the Moodle adapter.

LAMS Adapter

The LAMS adapter is developed and tested for the version 2.3 of LAMS. The code of this adapter is in Java, and it uses the Apache Struts and Spring frameworks. Besides, the access to the MySQL database is made through Hibernate libraries. The LAMS adapter also adds specific tables to this database in order to manage the specific information about *gluelets*. These tables follow a similar structure compared to the ones presented in the case of the Moodle adapter, and so, they are not replicated here. The Restlet framework for Java is employed to send the HTTP requests to the REST resources exposed by the GLUElet Manager.

The LAMS adapter implements the four use cases supported by the GLUE! architecture. Besides, the LAMS adapter has been developed to support the three main LAMS distinctive features according to educators [Bow11] and vendors [Dal07]. These features are: the monitoring of students' performance, the use of learning sequences and pathways, and the management of learning designs [Ala11b]. The support of these three features by the LAMS adapter is detailed next.

LAMS enables educators (also referred to as monitors in the LAMS terminology) to monitor students' performance in the monitoring environment, including also some built-in tracking features, such as time charts or sequence status. The GLUE! contract for VLEs requires that the list of VLE users sharing each tool instance, which may include the full course list or just the subset of students in a certain group, is sent in creation requests. Educators that assess students' work are also included in this list, as it happens in other VLE adapters too. Therefore, the LAMS adapter includes the educators delivering a course as extra users in every tool instance that is created. Once instances are actually created, educators can visualize their content and annotate

them, thus monitoring students' performance and giving them feedback. The remaining LAMS functionality for monitoring is also achieved just by following the LAMS contract.

LAMS enables the creation of sequences of activities in the authoring environment. On the other side, the GLUE! architecture independently manages each external tool and instance. The LAMS adapter takes advantage of this lack of correlation among tools and instances, in order to generate isolated activities that include external tools, and that may become sequences of activities through the LAMS own authoring logic for the management of transitions and branches. These sequences are supported by the LAMS adapter if students or educators manually decide to move forward or backwards. That happens in the LAMS monitoring environment when students click on the "*next activity*" button, or return to previous activities using the LAMS activity diagram. Likewise, branchings are supported if decisions are based on "*teacher's choice*" (the educator decides which group follows each branch) or "*group-based*" (each group follows a different branch). Nevertheless, the loosely-coupling of both the GLUE! architecture and its reference implementation precludes from using decisions based on "*learner's output*", since no mechanisms are implemented to retrieve outcomes from external tools. Few LAMS built-in tools, however, can take decisions based on this "*learner's output*" option.

LAMS was designed to facilitate the import/export of learning designs and their sharing among practitioners. Therefore, the LAMS adapter complies with the LAMS contract to allow the reuse of the same learning design in different LAMS courses. Designs including external tools can be imported and deployed in the same LAMS installation, as usual. The same applies to other LAMS installations, provided that they are connected to the same GLUElet Manager, or at least, to another GLUE! installation that integrates all the external tools included in the learning design.

Nevertheless, the current implementation of the LAMS adapter has two limitations compared to the Moodle adapter. First, the LAMS adapter does not support different configurations for different groups in the same LAMS activity. The reason is that the configuration of LAMS activities and the selection of the number of groups for each of these activities are independent actions in the LAMS authoring environment. This limitation also occurs with LAMS built-in tools, since LAMS only enables one single configuration for each activity. A simple solution to overcome this limitation is to use branchings in order to define different activities for each group; these activities can eventually be configured in different ways. The second limitation is that the functionality to reuse instances in different activities of the same LAMS lesson is not implemented in the current version of the LAMS adapter, although it is planned for future improvements. The usage of the LAMS adapter, as well as its installation and configuration processes are further detailed in sections 4.5 and 4.6.

MediaWiki Adapter

MediaWiki is not a VLE, as discussed in section 2.3.1, although it was designed to support collaboration and integration of external applications. Besides, MediaWiki is frequently used by practitioners as a centralized environment for the delivery of courses [Ala11a, Mar08a]. Significantly, MediaWiki meets the mandatory restrictions imposed in the GLUE! integration contract for VLEs. Therefore, the development of a VLE adapter for MediaWiki was considered feasible and convenient.

The MediaWiki adapter is developed and tested for the version 1.16 of MediaWiki. The code of this adapter is in PHP and JavaScript. PHP is employed for the processing and execution code, while JavaScript is employed for the generation of dynamic content in the graphical user interface. The MediaWiki adapter uses the cURL library for PHP to make HTTP requests to the REST resources exposed by the GLUElet Manager. Actually, the code for making requests and processing responses was reused from the one in the Moodle adapter. In a similar way, the code for processing configuration templates and representing them in the graphical interface was also reused.

The MediaWiki adapter adds a specific MySQL table to the MediaWiki database in order to manage the information about *gluelets*. Nevertheless, in this case, the number of fields in this table is much lower, since MediaWiki does not include features for the definition of courses, activities or groups. In fact, this table only stores a list of *gluelet identifiers*. Besides, MediaWiki pages (articles) locally store the *gluelet identifiers* for all the instances that are created within those pages too, under `<gluelet>` tags. These tags are generated by the own MediaWiki adapter, following the MediaWiki contract for the definition of new tags. These tags support one specific parameter, which is the title for the instance. Of course, additional text or resources can be included, as usual, in those MediaWiki pages in which instances are created. Despite the lack of support of courses, activities and groups in MediaWiki, educators may manually define courses by linking different articles, adding in each of these articles the activities that each group must carry out. Nevertheless, an important caution should be taken into account here, since every user registered in MediaWiki could access any instance. This limitation is a consequence of using MediaWiki as if it was a VLE, as explained in chapter 3, and is coherent with MediaWiki general policy of not using access control lists on a per-article basis. Remarkably, the MediaWiki adapter demonstrates that MediaWiki can integrate external tools using the GLUE! architecture, even though limited by its lack of pedagogical features for the definition and management of groups.

In any case, the MediaWiki adapter implements the four use cases supported by the GLUE! architecture. Besides, it easily supports the reuse of instances in different articles. The reason is that references to *gluelet identifiers* are stored in the MediaWiki database, but they are not associated to any specific article. Therefore, end-users can copy and paste the content of the

`<gluelet>` tag from one article to another. Interestingly, in the current MediaWiki adapter, the registration of new users in MediaWiki automatically launches a request for the update of users in external tool instances. The usage of the MediaWiki adapter, as well as the installation and configuration processes are further detailed in sections 4.5 and 4.6.

Other VLE Adapters

Other VLE adapters are being implemented or are planned to be implemented. For example, a VLE adapter for SharePoint LMS is currently under development, as part of a Spanish National project in partnership with the e-learning company *élogos*[12]. At the moment, a web part developed in ASP.NET is available for Microsoft SharePoint Server 2010. Nevertheless, this web part needs to be improved to incorporate the main Sharepoint LMS features in order to manage courses and users when integrating external tools. Besides, other VLE adapters are planned for Sakai and Drupal[13], which is another content management system that also supports collaboration and integration of external applications.

4.3.5. Tool Adapters

Nine tool adapters for Google Docs, MediaWiki, Dabbleboard, Apache Wookie, Doodle, Facebook Live Stream, Kaltura, Noteflight and any URL representing a web content are currently available in GLUE!-RI. All these adapters are called by the GLUElet Manager in order to create and configure, retrieve, update or delete external tool instances, as well as to retrieve the configuration templates of a given tool. After some of these calls, tool adapters need to interact with external tools to actually create (and configure), update or delete instances. Besides, tool adapters store some information about the instances they create to enable further actions (restricted to the four CRUD-like methods) over these instances. That information is locally stored in all the tool adapters developed, rather than in the own external tools, as a manner to reduce coupling. All this results in three commonalities of the available tool adapters.

- Tool adapters must meet the GLUE! integration contract for tools.
- Tool adapters must meet the integration contract imposed by the tools they wrap.
- Tool adapters must persist information about the instances they create.

The nine tool adapters have been developed in Java as REST services, employing the Restlet framework for Java. Therefore, these adapters expose the resources defined in section 3.4.2,

[12] `http://elogos.es`. Last visited: June 2012.
[13] `http://drupal.org`. Last visited: June 2012.

and support the four use cases detailed in section 3.5. Besides, these adapters need to persist some information about the created instances. This information is stored in a local file, rather than in a database. This implementation decision reduces the need for extra code in order to manage the access to databases, and also reduces the installation and configuration requirements for these adapters. The information stored for each external tool instance is represented in Table 4.5. Once again, this is just an option; different, and of course, more complex alternatives could be implemented to provide persistence, as long as they meet the GLUE! integration contract for tools.

Table 4.5: Information persisted in tool adapters.

Name	Purpose
Instance identifier	Integer that uniquely identifies the instance. This identifier is assigned to the REST resource exposed by the tool adapter to represent that instance under the form of `/instance/{instance_identifier}`.
Instance title	Title of the instance. Educators may set a title when configuring external tool instances. This title is stored here because it is a mandatory parameter that must be sent in Atom entries. If no title is set, then a predefined title is employed.
Instance date	Last modified date for the instance. That date can correspond to the creation date or to the last update of users. The date is stored here because it is also a mandatory parameter that must be sent in Atom entries.
Instance URL	URL representing the external tool instance. Significantly, some tools like Dabbleboard or Apache Wookie provide a specific URL for each user sharing the instance. In those cases, tool adapters store here a list of pairs with the username in the VLE and the URL of the instance. Afterwards, when retrieving instances, the "*callerUser*" parameter, which corresponds to the username that requests the instance in the VLE, is matched here in order to return the appropriate URL.

Current tool adapters

The nine available tool adapters at writing time are wrapping web-based tools, although the architecture is not restricted to this technology, as it is later discussed and exemplified. Besides, all these web-based tools provide REST-like interfaces, where some of the four well-known HTTP methods (i.e. `POST`, `GET`, `PUT` and `DELETE`) can be applied on different URIs. These kind of interfaces highly simplify the development of tool adapters, since they require very few lines of code for the communication with external tools, as it is demonstrated in the evaluation chapter.

The *Google Docs adapter* enables the creation, configuration (indicating a title and an initial file), and management of Google Documents, Google Spreadsheets, and Google Presentations instances. Each document, spreadsheet of presentation is considered a different instance. These tools define different contracts (e.g. Google Documents List API[14] and Google Spreadsheets API[15]) and support different configurations, although the interactions required for the creation

[14] `http://code.google.com/apis/documents`. Last visited: June 2012.
[15] `http://code.google.com/apis/spreadsheets`. Last visited: June 2012.

and management of tool instances are common, being they defined in the Google Documents List API. Despite the fact that this is a good example of a tool adapter that integrates several tools with similar contracts, other implementations of the Google Docs adapter could be developed to specifically integrate each of these three tools.

The *MediaWiki adapter* enables the creation, configuration (indicating a title and an initial content), and management of MediaWiki instances. Each MediaWiki page is considered a different instance. Although a VLE adapter was also developed to integrate external tools in MediaWiki, this platform is frequently employed as a collaborative text editor to generate and share content (e.g. Wikipedia), and so, it can be integrated as an external tool in VLEs to support collaborative writing activities.

The *Dabbleboard adapter* enables the creation and management of Dabbleboard instances. Each Dabbleboard drawing is considered as a different instance by this adapter. Significantly, Dabbleboard does not support the configuration of a drawing through its programmatic API[16]. The Dabbleboard adapter uses the list of VLE usernames, mapping them as Dabbleboard users in the Dabbleboard chat, indicating besides which of these users made the last change in the drawing.

The *Wookie widgets adapter* enables the creation and management of W3C widgets deployed in Apache Wookie servers. Therefore, this is a good example of GLUE! employed as a middleware architecture for other loosely-coupled integration approaches, as discussed in section 3.7.2. The concept of instance is defined by the own widgets, being that concept considered in Apache Wookie servers too. The current Wookie widgets adapter supports the configuration of some widgets like the YouDecide widget, which can be configured with an initial question and several predefined answers to choose from. VLE usernames are also mapped by this adapter in some widgets, such as the Natter chat.

The *Doodle adapter* enables the creation, configuration and management of Doodle instances. Each Doodle poll is considered a different instance. Besides, this adapter supports the initial configuration of Doodle polls by including the name of the poll, a description, and a set of possible answers.

The *Facebook Live Stream adapter* enables the creation, configuration and management of instances of this tool, which acts as a forum for Facebook users. This adapter allows to configure of the height and width of this forum when embedded in the VLE graphical interface. Besides, comments can be shared with everyone watching this instance, and also with end-users' friends on Facebook.

The *Kaltura adapter* enables the creation, configuration and management of Kaltura instances. Each Kaltura video is considered a different instance. This adapter supports the con-

[16] `http://dabbleboard.com/developer`. Last visited: June 2012.

figuration of an initial title, as well as the upload of a file in a video format, which actually represents the tool instance.

The *Noteflight adapter* enables the creation and management of Noteflight instances. Each Noteflight score is considered a different instance for this adapter. The Noteflight adapter does not support the initial configuration of Noteflight instances.

Finally, a tool adapter was implemented for the integration of web contents represented as URLs (*web content adapter*), even though VLEs usually enable the attachment of URLs as part of learning activities. Nevertheless, the web content adapter aims at overcoming the limitations found in VLEs like Moodle, regarding the management of groups when attaching URLs. More specifically, Moodle does not allow the use of groups when including URLs and other resources in learning activities. This precludes from particularizing the web content intended for each group in each activity. An example of such activities occurs when two groups must read different texts on the web with different points of view regarding a certain topic. Obviously, this adapter does not create, nor delete, instances, although the URL pointing at the web content must be set as a configuration parameter.

Other tool adapters

More tool adapters are being implemented or are planned to be implemented. For instance, a tool adapter for PiratePad[17], a web-based collaborative text editor, is under development. PiratePad is based on the open source and well-known EtherPad software[18], which was acquired by Google in 2009. Therefore, a tool adapter for PiratePad could also serve to wrap other similar tools based on the EtherPad software too, like PrimaryPad[19], TitanPad[20], or Sync.in[21]

Besides, other adapters are planned to be developed in collaboration with the owners. This is the case of the Video Learning Environment EVA3 (Educational Video with Analysis, Annotation & Assessment)[22], which was developed in the University of Sydney. EVA3 would be useful as an alternative to Kaltura, which is one of the external tools currently integrated.

Nevertheless, even though GLUE! can support the integration of tools developed with multiple technologies, all the aforementioned tools are web-based tools. A somewhat different example is the case of Synergo [Avo04], a Java standalone collaborative mapping environment developed by the University of Patras. A tool adapter for Synergo was partially implemented in collaboration with researchers from this University. This adapter wraps the Synergo Java

[17] `http://piratepad.net`. Last visited: June 2012.
[18] `http://code.google.com/p/etherpad`. Last visited: June 2012.
[19] `http://primarypad.com`. Last visited: June 2012.
[20] `http://titanpad.com`. Last visited: June 2012.
[21] `http://sync.in`. Last visited: June 2012.
[22] `http://sydney.edu.au/education_social_work/coco/research/projects/video`. Last visited: June 2012.

standalone tool, creating a JNLP (Java Network Launching Protocol) file [Jav11] with an instance of Synergo. This JNLP file is distributed within a web page that is locally stored in the tool adapter, returning its URL with the JNLP file attached, when learners request a Synergo instance within VLEs. The JNLP file is then downloaded and launched in a Java Web Start (JWS) container[23], which must be installed in end-users' systems. This is an important restriction for the integration of Java standalone tools within VLEs. Fortunately, JWS is included in the Java Runtime Environment, which is usually available in most computers and laptops. The Synergo adapter is not ready yet, since the personalization and configuration of instances have not been developed so far.

A similar approach, where tool adapters wrap, store and return the client of distributed applications within URLs could be useful to integrate tools developed with other technologies. For example, grid services like the DNSE [Bot10] or the Benchmarking Tool [Ala09] also provide Java standalone clients that could be wrapped and distributed into JNLP files. Tools with Java clients using other mechanisms, such as Java RMI[24] could be integrated under the same idea. Even applets could be easily added to HTML pages, which may be stored and distributed by tool adapters.

4.4. Developing new VLE and tool adapters

New adapters can be added to the existing ones by anybody interested in the integration of new tools and VLEs through the GLUE! architecture. VLE providers, tool providers, or any external developer may undertake the implementation of new adapters, or even the improvement of the existing ones. Those interested in contributing to the GLUE! architecture should read the GLUE! integration contracts for VLEs and tools, taking them as the theoretical grounding for the development of new adapters.

Particularly, the development of new tool adapters is supported by a special library called *GLUEcommon*, which was implemented for Java-based adapters. This library collects the code needed to expose the REST resources and methods defined in the GLUE! integration contract for tools. This library also gathers the code required to process the requests from the GLUElet Manager, to prepare the responses that must be returned, as well as to persist the information concerning the created instances. The process of implementing new tool adapters making use of the *GLUEcommon* library is well-defined, and typically requires only three steps. First, the configuration templates must be defined in XHTML files (XForms or HTML5), following the structure that can be found in the templates of other tool adapters. Second, the values of the template, once set by educators, must be retrieved following the code programmed for other

[23] `http://docs.oracle.com/javase/7/docs/technotes/guides/javaws`. Last visited: June 2012.
[24] `http://oracle.com/technetwork/java/javase/tech/index-jsp-136424.html`. Last visited: June 2012.

existing tool adapters. Then, three specific methods must be implemented: one for the creation and configuration of external tools; one for the update of users sharing external tools; and one for the deletion of external tools. These methods can be more or less complex depending on the specific tool contract. For instance, in non-web tools, the tool adapter is also responsible for the generation of URLs, as exemplified in section 4.3.5. These steps can be considered simpler compared to tighter integration approaches. Interestingly, the required code for the integration of external tools is similar to that in other approaches that also promote a loosely-coupled integration, as it is illustrated in the evaluation chapter. The Appendix C of this dissertation provides further information about the development of new tool adapters in Java using the *GLUEcommon* library.

The process of implementing new VLE adapters for GLUE!-RI is well-defined too, although it is much dependent on the specific VLE. Those implementing new VLE adapters are advised to start their developments from the existing extension modules that are normally given by the VLE providers. Then, developers need to map the educational concepts supported by VLEs (e.g. group, role, course, activity, etc.) with the concept of tool instances. After that, the requests to the GLUElet Manager must be added to the VLE adapter, taking into account the PHP or Java code from other existing VLE adapters. In any case, the use of a REST framework for these and other programming languages is highly recommended. Finally, the graphical user interface should be modified to include the selection of external tools and the configuration of instances by educators, as well as the subsequent visualization of these instances. These steps are more complex than in tool adapters, since they stem from the specific behavior of VLEs.

4.5. Installation and configuration of GLUE!-RI

The reference implementation of the GLUE! core and the available adapters are distributed in two different formats. The first format is a full package containing the GLUE! core, the three VLE adapters, and the nine tool adapters. The second format includes individual packages of the GLUE! core, and each of the adapters. In both cases, a binary distribution and a source code distribution are available for each of these packages. For the purposes of installing, configuring and using the code provided it is strongly recommended to download only the binary distribution. The manuals that must be followed in order to properly install and configure the reference implementation of the GLUE! core and the available adapters can be consulted in Appendix D, as well as in the `README.txt` file included in each of these packages.

It is noteworthy that the three available VLE adapters are installed as any other VLE module or extension. Therefore, VLE administrators just need to follow the documentation given by VLE providers for that purpose, in order to install VLE adapters. After that, VLE administrators need to configure two parameters for the communication between VLE adapters

and the GLUElet Manager. These parameters are the URI of the GLUElet Manager (to address the requests), and a timeout (to prevent VLEs from hanging up if external problems occur in a request). MediaWiki exceptionally requires a third parameter, which is a default username. This default username is employed as the "*callerUser*" when end-users do not need to register in a MediaWiki installation to edit its pages. All in all, the process of installing VLE adapters is pretty simple, since normally no extra knowledge is required to VLE administrators.

On the other side, the GLUElet Manager and the available tool adapters must be installed and configured following a slightly more complex process, which is detailed in the Appendix D of this dissertation for both Linux and Windows systems. The GLUElet Manager and these tool adapters run as independent services, using a port to listen to requests for the management of external tools. Cautions must be taken, because different element of the architecture installed in the same system must use different ports. Besides, the GLUElet Manager employs two databases for the management of information about the external tools (internal tool registry) and the instances created (*gluelets* repository), and so, a JDBC connector must be available in the system where the GLUElet Manager is installed. Details about how this JDBC connector must be set as part of the configuration of the GLUElet Manager are also explained in Appendix D. To facilitate the population of the internal tool registry, SQL scripts are provided in each of the packages that can be downloaded.

It is remarkable that the elements of GLUE!-RI can be offered by third-parties, except for VLE adapters that need to be installed together with VLEs. Therefore, the installation and configuration of the GLUElet Manager and the set of tool adapters is not always required, insofar as other providers running these elements may be used instead. This feature highly simplifies the setting process for VLE administrators that want their practitioners to use GLUE!-RI.

4.6. Usage of GLUE!-RI

The usage of GLUE!-RI is based on the four use cases detailed in section 3.5, as part of the overall behavior of the architecture. These four use cases are: the creation, configuration and assignment of external tool instances; the use of external tool instances; the update of users sharing external tool instances; and the deletion of external tools instances. Nevertheless, each VLE adapter and each tool adapter may provide its own implementation for these use cases. For example, the LAMS adapter requests educators the configurations for external tools within the authoring environment, although these configurations are not actually used until the creation of external tool instances in the monitoring environment. On the other side, some tool adapters, like those for Dabbleboard or Noteflight, do not support the configuration of external tool instances in their current implementations.

It is important to note that the available VLE adapters are intended to offer a seamless integration of external tools (as compared to built-in tools). Therefore, most of the steps followed for the management of external tools are common with those in built-in tools. For example, typical steps for the creation of external tools are as follows.

1. In the VLE menu where a list of built-in tools is shown to educators, a new option, namely *gluelet*, appears. Built-in tools are usually associated to learning activities, and so does the *gluelet* option. The first step for educators consists on selecting the *gluelet* option, instead of any other built-in tools.

2. The list with the available external tools that can be integrated as *gluelets* is then shown to educators. This step is obviously new, and it does not appear when selecting one built-in tool. The following step is the selection of one external tool by an educator.

3. The generic configuration of the tool (or activity) is requested to educators in one or more screens. This step is common with the case of built-in tools.

4. The specific configuration of tool instances is then requested to educators in one or more screens. This step also appears in most built-in tools. Significantly, in the case of Moodle, different configurations may be provided for each group, and even the reuse of instances from previous activities can be set in this step. Both features are new, and they are not offered with built-in tools.

5. External tool instances are created. These instances are automatically assigned to VLE users depending on their groups and roles. This step also happens with built-in tools.

Once the creation of external tools is accomplished, students logged in the VLE can perform the activities designed, by retrieving, visualizing and using their corresponding tool instances (either built-in and external), while educators may monitor their performance. Significantly, GLUE! aims at introducing minimum novelties in the instantiation and enactment of learning activities that require the integration of external tools.

GLUE!-RI has been tested by accessing VLEs from different web browsers. Most major browsers, like Firefox, Opera, Safari, or Chrome, do not present any problems to support this reference implementation. Nevertheless, Internet Explorer does not properly represent IFrames, and so, since URLs with external tools are normally embedded in IFrames, then the use of Internet Explorer with GLUE!-RI is not advised. A further explanation of the usage of GLUE!-RI can be consulted in Appendix E, where some screenshots exemplifying the usage of GLUE! within Moodle, LAMS and MediaWiki are provided. Additional videos with the usage of GLUE!-RI are also available at `http://www.youtube.com/user/gsicemic`. Besides, the steps for instantiating and enacting an authentic collaborative learning situation in Moodle and LAMS are documented in [Ala12c].

4.7. Conclusions

GLUE!-RI, the reference implementation of the GLUE! architecture, has been developed following the design decisions presented in the previous chapter. Remarkably, the examples of VLE and tool adapters included in GLUE!-RI respectively meet the GLUE! integration contracts for VLEs and the GLUE! integration contract for tools. Therefore, GLUE!-RI demonstrates that the architecture proposed in this dissertation can actually be implemented. Besides, it serves as a start up of VLE and tool adapters.

GLUE!-RI contains a reference implementation of the GLUE! core, three VLE adapters and nine tool adapters, although new VLE and tool adapters may be developed by anyone interested in contributing to the architecture. Besides, improvements on the elements distributed as part of GLUE!-RI can be made too, resulting sometimes in the support of new functionality (but taking into account the contracts defined by the architecture). As an example, the Moodle adapter was efficiently improved to promote the data flow among activities in which the same tool is employed, by supporting the reuse of instances. This new use case was easily implemented due to the loosely-coupling of the architecture.

GLUE!-RI is useful to demonstrate three important assertions made in this dissertation about the GLUE! architecture. The first one is that GLUE! can integrate external tools in other platforms where collaboration and integration are key issues, provided that these platforms meet the restrictions imposed in the GLUE! contract for VLEs. A clear example for that is the development of a VLE adapter for MediaWiki. The second assertion is that GLUE! can integrate external tools developed with multiple technologies, provided that these tools meet the restrictions imposed in the GLUE! contract for tools. Here, the example is the aforementioned integration of the Java standalone Synergo. The last one is that GLUE! can be used as a middleware architecture to achieve interoperability with other loosely-coupled integration approaches. The integration in VLEs of W3C widgets deployed in Apache Wookie servers through the GLUE! architecture exemplifies this assertion.

The process of developing new adapters has been simplified by providing the code of existing adapters as examples, and by detailing the steps that must be followed. Moreover, the *GLUEcommon* library is available for external developers. This library contains most of the code needed for the development of new adapters in Java, thus fostering the reuse of code. The installation and configuration of the elements distributed as part of GLUE!-RI have also been simplified, especially regarding VLE adapters, which are typically installed as any other VLE extension. Remarkably, the modularity of the architecture allows that different providers may offer different components of these elements. Thus, VLE administrators can easily configure their VLE adapters to use the GLUElet Manager (or some of the tool adapters) offered by external providers. The usage of GLUE!-RI requires a minimum extra knowledge to practitioners, since

the instantiation and enactment of collaborative learning situations that require integrated tools is very similar as compared to those situations in which only VLE built-in tools are employed.

Finally, it is noteworthy that the development of a reference implementation is useful to carry out the evaluation of the GLUE! architecture. The next chapter presents the methodology employed, the experiments realized, and the results obtained as part of this evaluation.

Chapter 5

Evaluation

The previous chapters have presented the design and implementation of the GLUE! architecture. This chapter delves into the evaluation of the proposed architecture. This evaluation has been carried out with the help of the reference implementation (GLUE!-RI), and is intended to assess whether GLUE! meets the stakeholders' requirements, and thus overcomes the limitations found in previous related works. Section 5.2 explains the methodology that has been followed for this purpose. Most of this evaluation is based on four authentic experiments, in which real educators and students used a different iteration of GLUE!-RI. Section 5.3 details the four experiments, which involve the instantiation and enactment of three collaborative learning situations that were designed by different educators in order to support their learning practices in several higher education courses. Section 5.4 discusses the compliance to the stakeholders' requirements with the data obtained from these experiments, distilling some conclusions about the GLUE! evaluation. Besides, this section also highlights other remarkable findings that came out during the realization of the experiments. Section 5.5 complements the evaluation by means of a comparison between GLUE! and other loosely-coupled integration approaches. This comparison analyzes the features they all provide for the integration of external tools in VLEs, and the development effort they demand to carry out this integration. Finally, the chapter ends in section 5.6 with the conclusions.

The evaluation of the GLUE! architecture, including the methodology employed, the four experiments, and the compliance to the main stakeholders' requirements has been published in [Ala12a]. More details about the findings obtained from the two experiments realized in the context of the Advanced Networking course can be found in [Ala12b]. Readers interested in the particular data collected during the evaluation process may consult them at `http://gsic.uva.es/glue`.

5.1. Introduction

The problem of integrating external tools in VLEs was discussed in chapter 2, identifying the requirements of the three main stakeholders interested in the accomplishment of that integration: practitioners, developers and VLE and tool providers. Besides, the main design issues that should be taken into account when tackling the integration problem were also identified

and analyzed in this chapter, pointing out the alternatives chosen by previous integrating works, as well as their limitations. These requirements, design issues and limitations were distilled as guidelines to propose a new integration architecture, called GLUE!, whose design and development were discussed and detailed in chapters 3 and 4. Nevertheless, in order to complete the research work, this integration architecture must be evaluated.

According to the overall research methodology followed in this dissertation [Gla95], which combines Adrion's engineering method [Adr93] and the empirical method [Big82], the evaluative phase must assess whether this architecture meets the objectives it was designed for; that is to say, the compliance to the stakeholders' requirements and the overcoming of the limitations found in previous related works. Besides, Glass [Gla95] suggests that the conclusions about the evaluation should stem from experimentation or observation, and might be achieved with the help of those systems or application developed for this purpose. Therefore, GLUE!-RI could be used by real practitioners in different experiments in order to obtain data and findings that support the evaluation of the GLUE! architecture. Significantly, since the objectives and contributions of this research belong to both software integration and CSCL domains, then both the technological and educational dimensions of this problem should be taken into account within these experiments [Zel02]. The particular methodology employed for the evaluation of the GLUE! architecture is presented next.

5.2. Evaluation methodology

The evaluation of the GLUE! architecture has been supported by the CSCL-EREM (Computer Supported Collaborative Learning Evaluand - oriented Responsive Evaluation Model) [Jor09] framework, which is especially indicated for the evaluation of systems and tools that promote collaborative learning. This framework allows to define an *evaluand*, and formalize multiple authentic experiments [Dew01] aimed at assessing this evaluand. Here, the evaluand is the compliance to the stakeholders' requirements by the GLUE! architecture. These requirements stem from different actors and cover different fields, and so, several methods and data sources were selected to obtain evidences and data about the compliance to these requirements during the experiments.

5.2.1. Evaluation framework and experiments

The CSCL-EREM is a framework that facilitates researchers and practitioners the formal evaluation of courses, resources, teaching strategies and software systems in CSCL settings [Jor09]. This framework is focused on an element, called *evaluand* (what is evaluated), and it is useful to define and formalize experiments whose purpose is to assess that evaluand. The

CSCL-EREM was the framework chosen to carry out the evaluation of this research work, since GLUE! is a software system intended for CSCL scenarios in which practitioners instantiate and enact different collaborative learning situation. In this case, the evaluand is defined as the compliance of GLUE! to the stakeholders' requirements.

The CSCL-EREM allows to classify the information about the different experiments into three different *facets*: *ground* (where the evaluand is evaluated), gathering the information about the context and the participants; *perspective* (why the evaluand is evaluated), covering the main goal pursued and other significant open questions; and *method* (how the evaluand is evaluated), indicating the data gathering techniques and the documents that support the conclusions. Figure 5.1 shows a generic representation of the CSCL-EREM framework including the evaluand (in the center of the figure) and the three different facets around the evaluand.

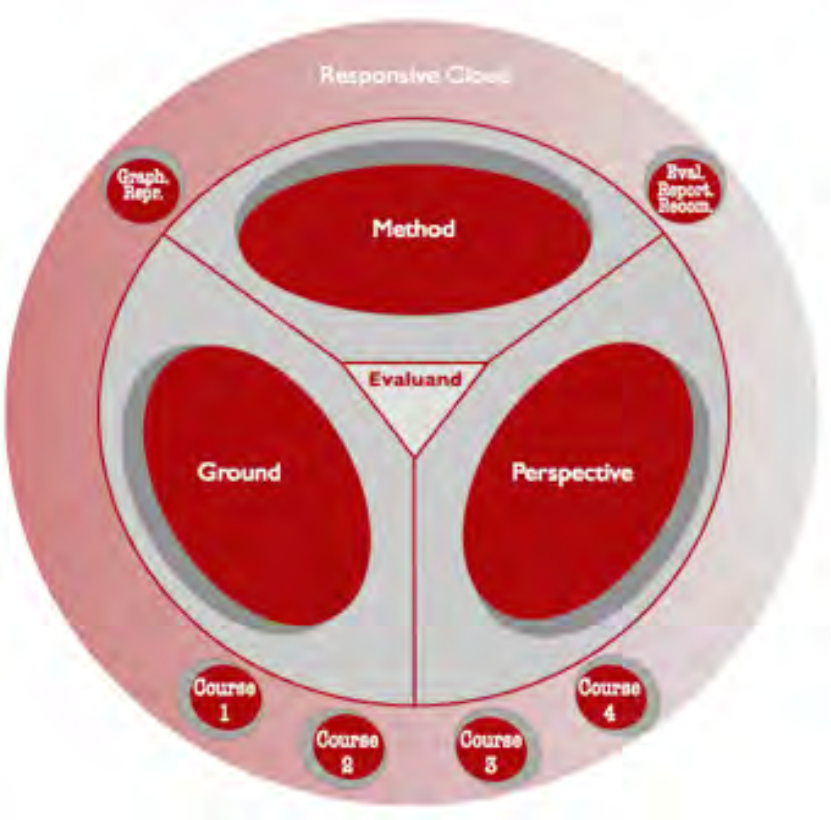

Figure 5.1: CSCL-EREM components. Figure taken from [Jor09].

Besides, the CSCL-EREM proposes four different *courses* (or itineraries), depending on the particular kind of evaluand [Jor09]: evaluation of CSCL programs, innovations or courses; evaluation of CSCL tools; evaluation of teaching strategies/learning resources to promote collaboration; and evaluation of CSCL projects. In this case, the second course (evaluation of CSCL tools) was found to be the most appropriate itinerary considering the evaluand defined

here. The CSCL-EREM also defines *recommendations to make evaluation reports* and a *graphical representation* to easily show all the information concerning a certain evaluation. Both the recommendations and the graphical representation were followed in order to organize and present the data obtained during this evaluation.

With the aim of assessing the evaluand (i.e. whether GLUE! complies with the stakeholders' requirements), multiple authentic experiments [Dew01] were defined. These experiments were formalized using the CSCL-EREM, and involved real end-users, like educators and students, and also developers, as suggested in [Dew01]. Significantly, the realization of authentic experiments involving different kinds of actors is a recurrently applied strategy for the evaluation of collaborative systems [Bot08,Iso10]. The need of involving human subjects entails an additional difficulty because the opportunities to organize and set authentic experiments are normally quite limited. For example, in this evaluation the experiments were carried out during several face-to-face and remote activities performed as part of different higher education courses, and so, the timeline of the evaluation had to be adapted to the program and schedules of these courses.

Particularly, four main experiments instantiating and enacting three different collaborative learning situations were carried out within three different courses (more details about these experiments and situations are later given in section 5.3). Besides, two secondary experiments without real practitioners were performed in order to replicate the instantiation processes in the four main experiments with and without GLUE!. Apart from providing evaluation results, these experiments also served to receive useful feedback from educators and students; this feedback was employed to improve the design and development of the proposed architecture. This idea perfectly fits within the overall research methodology [Gla95], which is iteratively developed, as it is the implementation of GLUE!-RI. The use of authentic experiments is not without problems though, insofar as the replicability of the experiments is rather limited, and results may be biased by specific participants. Nevertheless these experiments show that the GLUE! architecture can be used by real educators and students to support learning activities, and allow to detect the benefits and limitations of using this architecture. Here, it is convenient to note that most integration works, such as Apache Wookie [Wil08] or Basic LTI [IMS10b], do not provide evaluation results of their use by real practitioners, even though these approaches, like GLUE!, are intended to solve an important research problem in the education field.

Finally, it is important to remark that the stakeholders' requirements, and by extension the evaluand, cover both the technological and educational fields, as already discussed. Thus, several methods and data gathering techniques were combined to collect and analyze the data obtained from these experiments.

5.2.2. Evaluation methods and data sources

Different methods and data sources were employed to assess the compliance to each of the six stakeholders' requirements, and to compare GLUE! and other related works. Table 5.1 summarizes these requirements, as well as the evaluation methods and data sources employed to assess each of them.

Table 5.1: Stakeholders' requirements, methods and data sources employed for their evaluation.

Stakeholder	Tag	Requirement	Evaluation methods	Data sources
Practitioners	**REQ1**	GLUE! should enable the instantiation of individual and collaborative activities that require the integration of external tools with an attainable effort for educators.	Multiple experiments; mixed method	Likert scales; open text questionnaires; interviews; time and complexity measurements
	REQ2	GLUE! should enable the enactment of collaborative activities that require the integration of external tools, facilitating the collaboration among participants.	Multiple experiments; mixed method	Likert scales; open text questionnaires; focus groups
	REQ3	GLUE! should support the integration of existing and popular VLEs and tools.	Multiple experiments; feature analysis (formal experiment)	Reference implementation of GLUE!
	REQ4	GLUE! should support the integration of many external tools.	Multiple experiments; feature analysis (formal experiment)	Reference implementation of GLUE!
Developers	**REQ5**	GLUE! should demand an attainable development effort for the integration of new tools and VLEs.	Multiple experiments; new SLOC and time invested	Reference implementation of GLUE!
VLE and tool providers	**REQ6**	GLUE! should be built over existing VLEs and tools, rather than modifying their implementations.	Multiple experiments; feature analysis (screening mode)	Reference implementation of GLUE!

Findings concerning **REQ1** and **REQ2** were obtained from the experiments, using the particular *mixed method* proposed in [Mar03]. This mixed method has successfully been employed for the educational evaluation of other systems that also promote collaboration [Bot08, Nav11].

The first step, according to this mixed method, is the creation of a set of categories with the different concepts to be analyzed in the experiments. For the assessment of REQ1 and REQ2 these categories were: the previous experience of end-users with the VLE and the tools; the easiness of use of the VLE and the tools; the usefulness of the VLE and the tools to support the proposed activities; the support of the overall system to the collaboration and groupwork; the adequation of the experiment to the course program; the problems found; and the suggestions for improvement.

The second step is the collection of data. According to this method, data from different quantitative and qualitative sources should be obtained to reinforce the conclusions. In this case, quantitative data were collected from optional Likert scales [Lik32] passed to the participants,

while qualitative data came from open text questionnaires in optional forms filled out by the participants, individual interviews with the educators, focus groups with the students [Mor98], monitoring reports of the students' performance, and observations of the instantiation and enactment processes. Besides, additional quantitative data from the instantiation time and complexity were harvested using the two secondary experiments replicating the instantiation of the activities with and without GLUE!, in order to back up the conclusions concerning the first requirement.

Finally, in the last step data from these varied sources should be triangulated as suggested in [Mar03]. Significantly, quantitative techniques do not aim at demonstrating hypothesis, but at gaining insight and detecting tendencies, which are afterwards confirmed or discarded using qualitative techniques. Examples of the triangulation of quantitative and qualitative data sources are later given in section 5.4, when discussing the findings for REQ1 and REQ2.

REQ3 and **REQ4** were researched using a *feature analysis method* in its *formal experiment approach* [Kit97b]. A feature analysis is a qualitative evaluation method, in which the requirements that users expect for a particular situation are mapped to those features that a tool or system (intended to support that situation) should possess [Kit96b]. In the formal experiment approach, these features are evaluated by means of experiments with end-users. That helps increasing the confidence in the results, and also reducing the risk of systematic bias; however, as in any feature analysis, obtained results still have a certain degree of subjectivity [Kit97a]. The four aforementioned experiments and the reference implementation of GLUE! supported the discussion concerning REQ3 and REQ4.

The compliance to **REQ6** was also researched by means of a *feature analysis method*, but this time in its *screening mode approach*. This kind of feature analysis is performed by one person, and so, the evaluation criteria are subjective. Nevertheless, this approach is particularly suitable to survey a large number of software components, being thus very useful to screen all the available VLE and tool adapters in GLUE!-RI. Apart from the code of those VLE and tool adapters employed in the four authentic experiments, the code of the remaining available elements in GLUE!-RI was also surveyed. Besides, this kind of feature analysis was also employed to compare the contracts of GLUE! and those in other loosely-coupled approaches.

Finally, the satisfaction for **REQ5** was assessed by studying the *incremental development effort* required to enact the four experiments supporting this evaluation. Two quantitative metrics that are commonly used as indicators of software complexity [Pen07] were employed to obtain indirect measures of the effort required to integrate new external tools and VLEs with GLUE!. These metrics are the *new source lines of code* (SLOC) that must be developed to accomplish a certain integration [Alb83], and the *time invested* in that task. The code from GLUE!-RI was analyzed using these two metrics, as well as compared to that in other loosely-coupled approaches. These metrics provide simple empirical estimations of quantitative relationships among efforts, although cautions should be taken because they do not address issues

like language difficulty or developers' skills. Therefore, if different languages and developers are involved, only gross differences (e.g. changes in the order of magnitude) should be considered in order to draw conclusions on these data. Other classical metrics of software complexity, such as Function Points (FP) [Alb83] or Use Case Points (UCP) [Kus04] were discarded here, since they do not consider the amount of code reused, which is significantly higher in GLUE!, due to the promotion of a many-to-many approach and the use of well-defined loosely-coupled integration contracts based on popular standards.

5.3. Collaborative learning situations

Three authentic collaborative learning situations were instantiated and enacted in four occasions by different practitioners in higher education courses, being these the four experiments supporting most of the GLUE! evaluation. The three situations were designed covering various knowledge domains, collaborative strategies and durations. Besides, different VLEs and external tools were employed in the four experiments associated to these three situations. Interestingly, each of these experiments also served to test a different GLUE!-RI iteration (see section 4.3.2) in a real setting, thus helping to the incremental improvement of the architecture and its implemented elements. Tables 5.2 and 5.3 respectively summarize the three collaborative learning situations designed, and the four experiments that were carried out for the GLUE! evaluation.

Table 5.2: Summary of the three collaborative learning situations.

Collaborative learning situation	**Situation I**	**Situation II**	**Situation III**
Course	Advanced Networking (AN)	Software Engineering (SE)	Information and Communication Technologies Applied to Education (ICTE)
Degree	Telecommunication Engineering	Telecommunication Engineering	Early Childhood Education
Year	Third out of five	Third out of five	First out of four
Kind of situation	blended collaborative learning situation	face-to-face collaborative learning situation	blended collaborative learning situation
Main learning objective	Reflect and discuss about the design of a message server	Review and discuss the main concepts in UML class diagrams	Analyze and discuss the role of technology in Spanish schools
Number and kind of activities	5 collaborative activities	1 individual activity; 2 collaborative activities	1 individual activity; 4 collaborative activities
Duration	1 week	3 hours	1 week
Collaborative learning flow pattern	pyramid	pyramid	no pattern

Table 5.3: Summary of the four authentic experiments.

Tag	**AN-2010**	**SE-2011**	**AN-2011**	**ICTE-2012**
Collaborative learning situation	Situation I	Situation II	Situation I	Situation III
Date	November 2010	May 2011	November 2011	February 2012
Number of educators	2 (technological background)	1 (technological background)	2 (technological background)	1 (pedagogical background)
Number of students	47	10	51	25
Group settings	24 pairs (1-2 students); 7 supergroups (6-8 students)	2 groups of five students	28 pairs (1-2 students); 8 supergroups (6-8 students)	12 pairs (2-3 students); 5 supergroups (5 students)
VLE	Moodle	MediaWiki	Moodle	LAMS
Built-in tools	-	-	-	forum, mind map
External tools (instances)	Dabbleboard (31); Google Documents (24); Google Presentations (7)	Google Documents (12); You Decide W3C widget (40)	Dabbleboard (36); Google Documents (28); Google Presentations (8)	Google Presentations (5); Doodle (1)
GLUE!-RI iteration	1	2	3	4

5.3.1. Collaborative learning situation I

Advanced Networking (AN) is a third-year course (out of five) in Telecommunication Engineering at the University of Valladolid, Spain. One of the main AN objectives is to get students to study, understand and use several alternatives for the development of distributed applications, complementing also the overview of telecommunication services and networks acquired in previous telematics courses. In one of the core practical exercises, AN students must develop a message server, as a representative example of a distributed application in the network layer of computer networking. Before the students start the generation of code, educators want them to reflect and discuss about how the message server should be designed. AN educators designed a blended collaborative learning situation to help students to achieve these objectives. This situation follows a well-known collaborative learning flow pattern called *pyramid* [Her06a], which favors reaching a gradual consensus among students working on the same leaning objectives.

The situation includes five collaborative activities in two levels of the pyramid. In the first level, students draw the time sequence diagram and the flowchart of the message server in pairs, justifying the main decisions taken. A shared whiteboard and a collaborative text editor could be provided to support these activities. Then, students are gathered in supergroups (groups of 6-8 participants), and review the diagrams of the other pairs belonging to their supergroup, by means of, for instance, a collaborative text editor. Finally, each supergroup agrees on the final diagrams and makes a presentation explaining the challenges and conclusions of the experiment. A shared whiteboard and a collaborative presentation tool may serve for this purpose.

The situation designed for AN was instantiated and enacted in two different experiments in consecutive years: 2010 (**AN-2010**) and 2011 (**AN-2011**) [Ala12b]. The technological support for this situation was Moodle in both experiments, since this was the institutional VLE at the University of Valladolid. This VLE natively supports the instantiation and enactment of collaborative learning situations by creating and managing users, groups and learning activities arranged in Moodle courses. Nevertheless, Moodle built-in tools do not include shared whiteboards, nor presentation tools, while collaborative text editors (e.g. wiki tool) are quite limited. So, AN educators employed GLUE! to integrate some external tools that could support the aforementioned activities. In both experiments the integrated tools were Dabbleboard, Google Documents and Google Presentations. The first GLUE!-RI iteration was used in AN-2010, while the third GLUE!-RI iteration supported AN-2011. Figure 5.2 depicts the CSCL-EREM diagram for the AN-2011 experiment; the only changes between the diagrams in AN-2010 and AN-2011 were the number of students (47 and 51 in the given order) and the date in which the happenings occurred (face-to-face sessions). Further details on the number of groups, and external tool instances created within these two experiments are summarized in Table 5.3.

5.3.2. Collaborative learning situation II

Software Engineering (SE) is a third-year course (out of five) in Telecommunication Engineering at the University of Valladolid. Students participating in SE learn that developing software is a very complex process that involves many other activities apart from writing code. Educators delivering SE encourage students to work in medium-size groups (5 members) throughout the course, since the development of software is generally a collaborative process that may involve people playing different roles (e.g. commercial, analyst, developer, etc.). During the SE laboratory assignments, students work in the capture of requirements, analysis, design and implementation of a realistic software project, following the Unified Process (UP) [Lar02], and generating, among other artifacts, UML (Unified Modeling Language) diagrams [Boo99] (e.g. use case diagrams, class diagrams, sequence diagrams, etc.). After finishing the project, the last two sessions of SE serve to summarize the whole course, and to prepare the students for the final exam. One recurrent exercise in these exams is a true or false quiz about a given class diagram. This kind of exercise helps students review the main concepts of object oriented programming, such as inheritance, overriding, or polymorphism [Lar02]. In this context, a face-to-face collaborative learning situation that also employs the pyramid pattern with three levels is designed to help students reflect on this kind of exercise.

This situation involves one individual and two collaborative activities. In the first level of the pyramid, students individually answer a twenty-question quiz about a given class diagram, with the aid of, for instance, a poll tool (to collect responses) and a text editor (to write justifications). In the second level, students are arranged in groups of five, see the responses given by

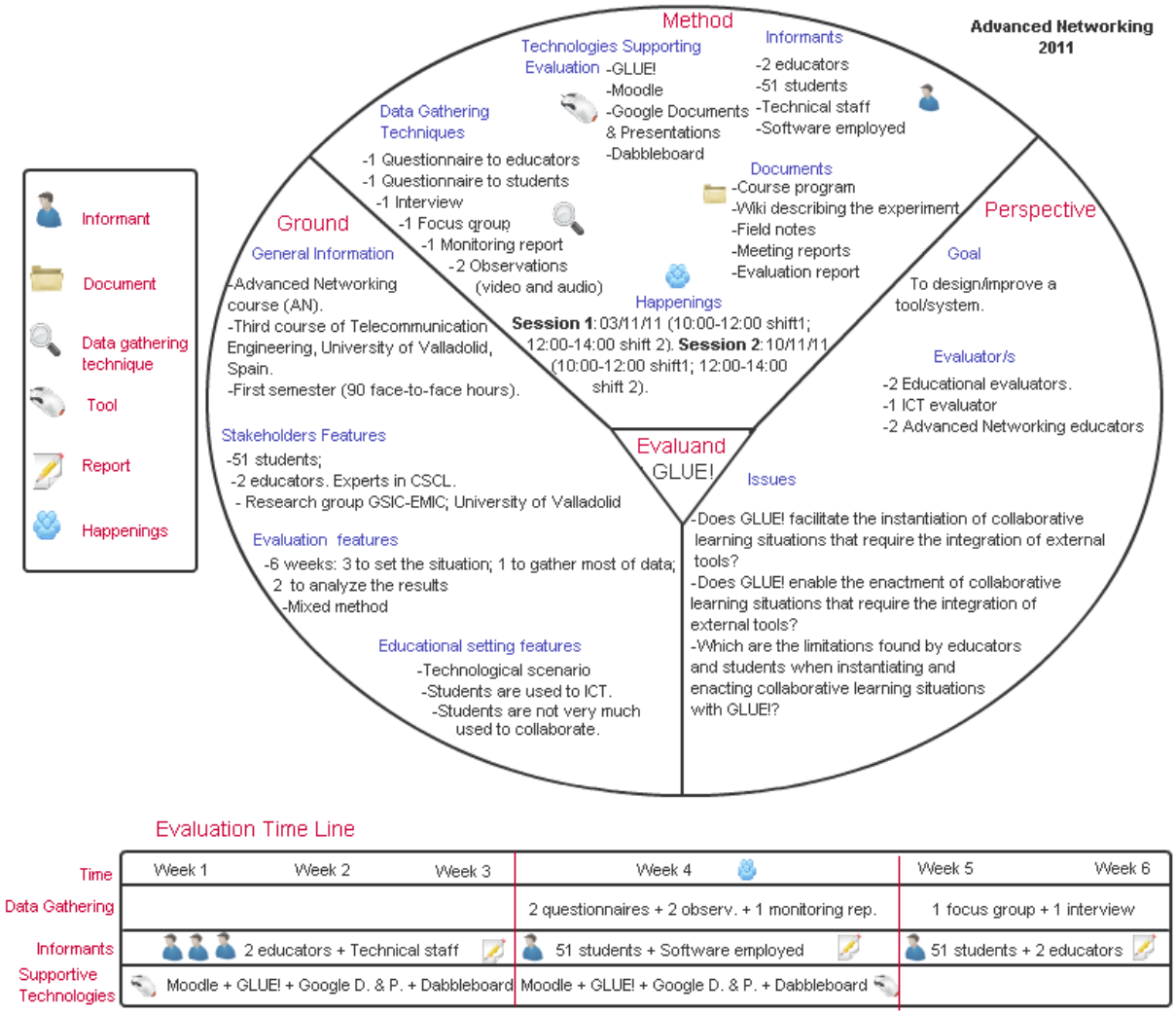

Figure 5.2: CSCL-EREM representation diagram for the GLUE! evaluation in AN-2011.

their group partners, and agree on a single response for each of the questions. The same tools would be needed, although here the text editor should be collaborative. The third stage occurs at class level, where a final consensus is reached mediated by the educator.

This situation was enacted in 2011 (**SE-2011**) using MediaWiki as a learning platform (see Table 5.3). MediaWiki was the learning platform employed throughout the SE course, and so, educators also chose it for this experiment. Unlike Moodle, MediaWiki does not support the management of groups and activities, although they can be manually mapped in the form of different MediaWiki pages (each page for each activity of each group). Besides, MediaWiki includes very few built-in tools, and so, SE educators used GLUE! to integrate Google Documents (as a collaborative text editor) and the You Decide W3C widget (as a poll tool). The CSCL-EREM diagram representing the experiment is shown in Figure 5.3.

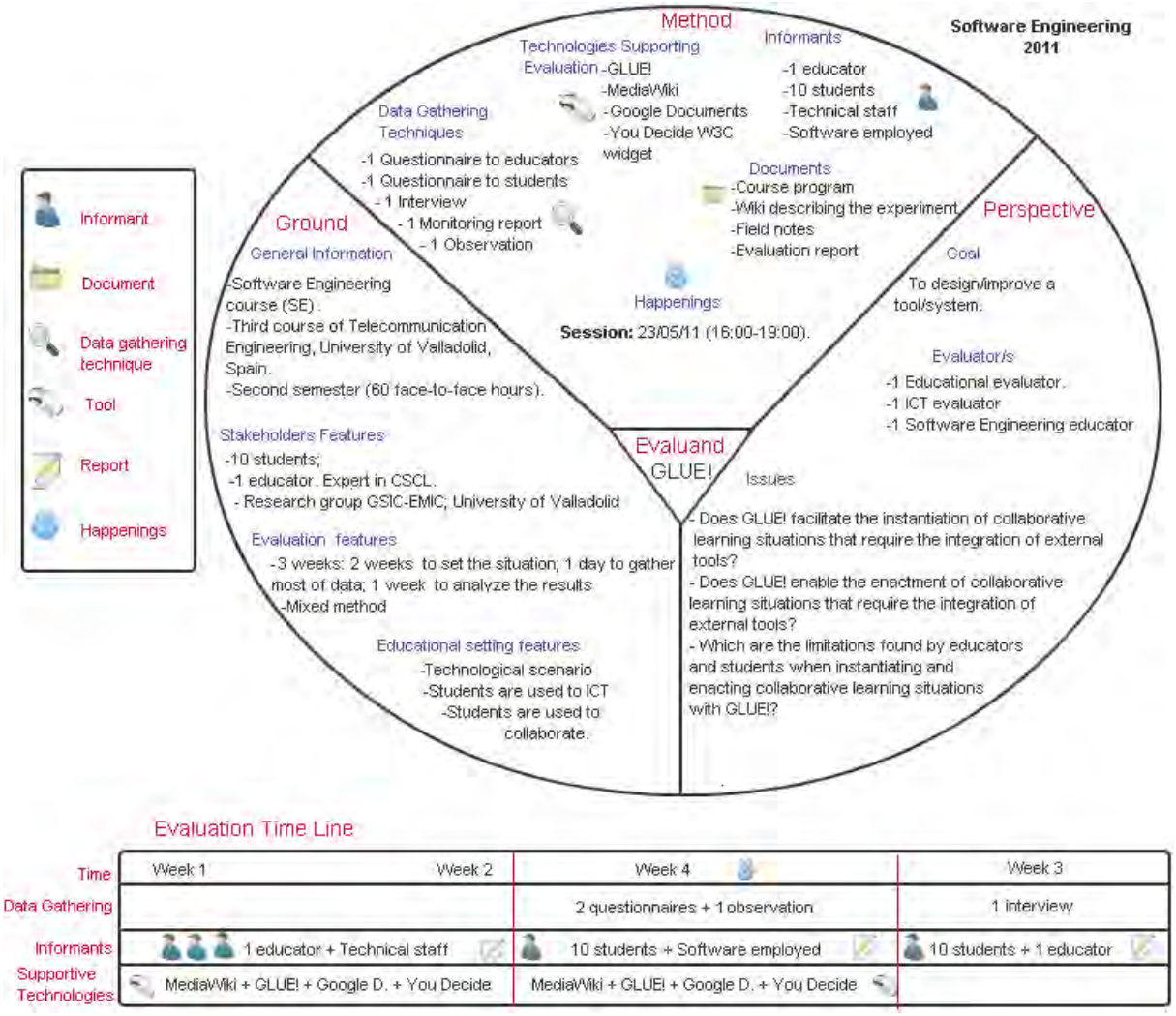

Figure 5.3: CSCL-EREM representation diagram for the GLUE! evaluation in SE-2011.

5.3.3. Collaborative learning situation III

Information and Communication Technologies applied to Education (ICTE) is a first-year course (out of four) in the Degree of Early Childhood Education at the University of Valladolid. The main purpose of this course is to introduce future educators in new technologies and media. Therefore, this course covers the role of ICT in the current society, exemplified through the use of several software tools and learning platforms, such as blogs, wikis, social networks and VLEs. In one of the practical exercises, ICTE students must study and discuss the role that technology currently plays in Spanish schools. In order to help students discuss about this subject, a blended collaborative learning situation is designed for this exercise.

This situation includes a series of individual and collaborative activities that need to be enacted in a certain order. First, students individually read two official online reports about this

subject. Then, they answer a brief questionnaire in pairs; a communication tool may be provided for this activity. After that, they are arranged in groups of five, in order to organize the concepts extracted from the previous reports using, for instance, a collaborative concept map tool. Then, working in the same groups of five, they make a presentation containing the conclusions of their work; this activity could be supported by a collaborative presentation tool. Finally, students individually vote the best of the presentations; a voting tool might serve for this purpose.

This situation was enacted in 2012 (**ICTE-2012**) using LAMS (see Table 5.3). The experiment, represented in the CSCL-EREM depicted in Figure 5.4, mixed built-in and external tools. The LAMS built-in forum and mind map tool were employed for the first two activities. Nevertheless, LAMS does not include presentation nor voting tools, and so, Google Presentations and Doodle were integrated using GLUE! for the last two activities. Significantly, ICTE students used several VLEs during the course to perform their regular activities, so that they also get trained on the usage of VLEs, and develop an opinion on several alternatives (useful when they become teachers); it is relevant to note that this was the first time they worked with LAMS.

5.4. Compliance to the requirements

The four experiments were instantiated and enacted by real educators and students between November 2010 and February 2012. Findings from these experiments were collected to show whether the GLUE! architecture meets the stakeholders' requirements (see Table 5.1). For a better understanding, these findings are organized following the sequence of requirements from REQ1 to REQ6. Besides, some other interesting findings that came out during the instantiation and enactment of these experiments are discussed at the end of this section.

5.4.1. Instantiation of individual and collaborative activities (REQ1)

The fact that the different collaborative learning situations presented in section 5.3 could be actually put into practice demonstrates that GLUE! allowed different educators to instantiate their desired individual and collaborative activities. In order to assess whether the effort put by these educators was attainable, the approximate instantiation time with and without GLUE! was employed as an indirect measure of that effort.

Table 5.4 shows the approximate instantiation time employed for the creation, configuration and assignment of the external tool instances required in the four experiments; supplementary details are provided for AN-2011, since this is the experiment in which a higher number of instances were managed. ICTE-2012 is a good example in which educators obtained a high benefit by using GLUE!, saving an 84,6% of the instantiation time in the LAMS authoring environment; the deployment of the lesson in the LAMS monitoring environment, however, was accomplished

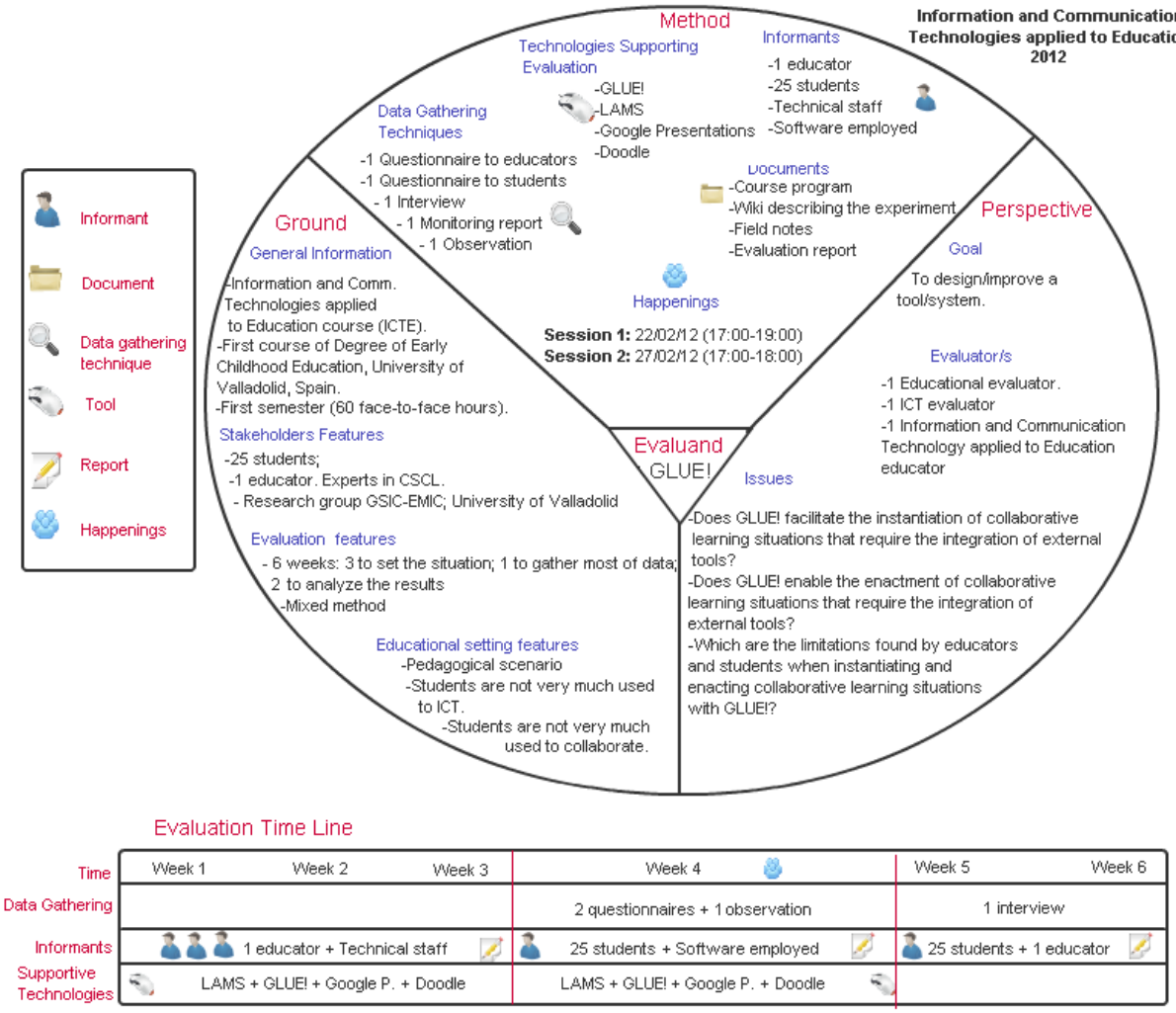

Figure 5.4: CSCL-EREM representation diagram for the GLUE! evaluation in ICTE-2012.

in less than in minute both with and without GLUE!. The saving was also representative in SE-2011, where using GLUE! took a 35% less of the instantiation time. The lesser saving was due to the fact that MediaWiki does not support the management of groups and activities, so that even with GLUE!, instances for different groups must be created in different steps.

Besides, it can be seen that in those cases where many external tool instances had to be created (AN-2010 and AN-2011), a great reduction in the instantiation time was obtained (more than 82%). The case of Google Documents is quite representative, since the 28 instances were created, configured and assigned at once within the same Moodle activity thanks to GLUE!, thus reducing a lot the effort and the time (about 93%). Google Presentations supported one of the activities in the supergroup level of the pyramid, being the saving lower as compared to the case of Google Documents, but still very significant (more than 71%). This is due to the fact that learning activities involving supergroups require the creation of a different Moodle activity

Table 5.4: Approximate instantiation time for the creation, configuration and assignment of external tool instances with and without GLUE! in the four experiments, with details for AN-2011.

Experiment	*Number of external tool instances*	*Time with GLUE! (minutes)*	*Time without GLUE! (minutes)*
ICTE-2012	**6**	**1**	**6.5**
SE-2011	**12**	**4.5**	**7**
AN-2010	**62**	**6.5**	**37.5**
AN-2011	**72**	**7.5**	**42.5**
Dabbleboard instances	36	4	13.5
Google Documents instances	28	1.5	22
Google Presentation instances	8	2	7

for each of them. Interestingly, the time needed to delete external tool instances, although less significant, also presented better results with GLUE! (e.g. 37 seconds to delete the 28 Google Documents instances with GLUE! versus about 1.5 minutes without GLUE!).

It is important to point out that the instantiation of these four experiments without GLUE! was feasible because the tools employed were all web applications whose instances could be represented as URLs. Thus, for each tool, new instances could be created using its web graphical interface, and then these URLs could be copied and pasted in some of the VLE activities. The You Decide W3C widget (employed in SE-2011) was an exception to this, and despite being a web tool, different instances of this widget could not be created directly using the Apache Wookie graphical interface.

Remarkably, GLUE! not only reduces the instantiation time, but also the instantiation complexity, since the creation, configuration and assignment of external tool instances can semi-automatically be made as part of a single VLE activity (i.e. the educator decides when to create and configure instances within the VLE interface, but these instances are transparently assigned to VLE groups). As an example, with GLUE! the creation, configuration and assignment of the 28 Google Documents instances in AN-2011 for the predefined groups in the first level of the pyramid demanded educators only 10 interactions with the Moodle interface: to add a new Moodle activity, namely *gluelet*; to set a name for the activity; to set a description for the activity; to select the external tool (Google Documents) from a list; to select the group mode ("*separate groups*"); to click on the "*save and continue*" button; to set the title for the Google Documents instances; to click on "*upload file*"; to actually select an initial file; and to click on the "*apply to all*" button. Extended details about the instantiation process of this collaborative learning situation with GLUE! are provided in [Ala12c]. In contrast, without GLUE!, 11 interactions with Google Documents (two to upload a file, three to set a title, five to change the sharing permissions, and one to copy the URL) and 6 interactions with Moodle (to add a new "*link to a*

file or web site" resource; to set the name; to set the description; to paste the URL; to select a grouping; and to click on "*save and display*"), were required for each of the 28 instances, yielding a total of 476 interactions. Therefore, without GLUE!, the instantiation of this collaborative activity becomes a very cumbersome and error prone process.

To contrast this quantitative data with educators' perceptions, results from the questionnaires filled out by the educators were analyzed (see Table 5.5), discovering that 3 out of 4 agreed or completely agreed that the effort they needed to instantiate the learning activities with GLUE! was attainable. The other, which was an AN educator, somewhat agreed on this statement. Interestingly, they all agreed or completely agreed that the benefit obtained was worth the effort. Moreover, they all agreed or completely agreed that GLUE! facilitated the instantiation of their collaborative learning situation.

Table 5.5: Aggregated answers obtained from the educators that instantiated the experiments. Statement 1 (S1): "*The instantiation effort is attainable*"; Statement 2 (S2): "*The benefit obtained was worth the instantiation effort*"; Statement 3 (S3): "*GLUE! facilitated the instantiation of the collaborative learning situation.*"

Options	*S1*	*S2*	*S3*
Completely agree	1/4 (25%)	1/4 (25%)	**2/4 (50%)**
Agree	**2/4 (50%)**	**3/4 (75%)**	**2/4 (50%)**
Somewhat agree	1/4 (25%)	0/4 (0%)	0/4 (0%)
Somewhat disagree	0/4 (0%)	0/4 (0%)	0/4 (0%)
Disagree	0/4 (0%)	0/4 (0%)	0/4 (0%)
Completely disagree	0/4 (0%)	0/4 (0%)	0/4 (0%)
No answer	0/4 (0%)	0/4 (0%)	0/4 (0%)

Interviews with the educators supported these results, and also served to discover that the AN educator that somewhat agreed with the first statement found some difficulties when creating new activities in Moodle that were due to the built-in management of group settings (groups and groupings) in this VLE. Though this problem can be attributed to Moodle, it affects the performance of GLUE!, which relies on the specific usage of the group settings in each VLE to facilitate the instantiation of collaborative learning situations. Actions taken to tackle this problem included enforcing clarifications in the GLUE! user manuals regarding the usage of group settings in this and other VLEs, and the generation of supplementary videos illustrating the steps for instantiating different learning activities[1].

All in all, evaluation results showed that GLUE! enabled the instantiation of these four experiments, demanding an attainable effort to educators. Analogous results could be expected in other collaborative learning situations with activities of similar complexity. Thus, it can

[1] `http://www.youtube.com/user/gsicemic`. Last visited: June 2012.

be concluded that GLUE! meets **REQ1**. The compliance to this requirement stems from the support of the life cycle of external tools within VLEs. Even though sometimes the instantiation of individual and collaborative activities could be made without GLUE!, the time and complexity that process would require may easily become unattainable, especially as the number of groups, activities and tools grows, as illustrated with examples in this section.

5.4.2. Enactment of collaborative activities (REQ2)

The four experiments were enacted by 133 real students in total with different backgrounds on engineering and pedagogy, although only some of them decided to answer the optional questionnaires and participate in the focus group. Data collected is still very significant because those with positive or negative opinions are more likely to participate in research studies, thus stressing the benefits and limitations of the use of GLUE! during the experiments. Table 5.6 collects the aggregated results from those questions showing evidences of the support of GLUE! to collaboration and groupwork.

Table 5.6: Aggregated answers obtained from the students that enacted the experiments. Question 1 (Q1): "*How much did you collaborate with your partners?*"; Question 2 (Q2): "*How much did the technological support facilitate the performance of the activities in collaboration with your partners?*"; Question 3 (Q3): "*How easy/difficult was to see your partners' contributions along the activities?*"

Options	*Q1*	*Q2*	*Options*	*Q3*
Very much	11/61 (18.03%)	**24/61 (39.34%)**	*Very easy*	19/61 (31.15%)
Much	**44/61 (72.13%)**	23/61 (37.70%)	*Easy*	**25/61 (40.98%)**
Some	5/61 (8.20%)	10/61 (16.40%)	*A bit easy*	12/61 (19.67%)
A Little	0/61 (0%)	3/61 (4.92%)	*A bit difficult*	2/61 (3.28%)
Little	0/61 (0%)	0/61 (0%)	*Difficult*	0/61 (0%)
Very little	0/61 (0%)	1/61 (1.64%)	*Very difficult*	2/61 (3.28%)
No answer	1/61 (1.64%)	0/61 (0%)	*No answer*	1/61 (1.64%)

Results to these questions show that: 90.16% of students thought that they collaborated *much* or *very much* with their partners during the proposed activities; 77.04% considered that the technological support (which included a VLE, some built-in tools –only in ICTE-2012–, some external tools, and GLUE!) facilitated *much* or *very much* this collaboration; and 72.13% agreed that seeing the contributions from their partners was *easy* or *very easy*. Interestingly, they all *agreed* or *completely agreed* that having all the tools integrated in a single graphical interface facilitated the enactment of the proposed activities. This agreement was particularly remarkable in those students without a technical background (ICTE-2012).

These results were confirmed by comments from students (in the open text questions and in the focus group), which also highlighted the benefits of having the necessary tools integrated in the VLE, and the usefulness of the whole technological support to facilitate the collaboration and groupwork: "*Everything is integrated in the platform in a convenient and very simple way*" (ICTE-2012); "*It saves time because everything is integrated in the platform*" (SE-2011); "*It is fantastic to collaborate because you can easily see the answers of your group partners*" (ICTE-2012); "*It was very easy to see the contributions of my group partners, just by logging into Moodle*" (AN-2010); "*I think it [the performance of the activities in collaboration] was facilitated because of the ease with which we could share and visualize our partners' work*" (AN-2011).

These comments also served to detect that the negative answers in Table 5.6 mainly came from a pair of students enrolled in the AN-2010 course, who were not able to find the built-in Moodle option to visualize the instances of their supergroup partners in the review activity: "*We did not see our partners' work because we did not know there was an option to change the group displayed*". Besides, some other negative comments were due to the particular tools employed to carry out the learning activities. For example, some AN-2010 and AN-2011 students complained about the limitations of Dabbleboard for drawing the sequence diagram and the flowchart of the message server (e.g. "*the size of the canvas in the shared whiteboard was sometimes insufficient to draw the diagrams*"). Besides, some ICTE-2012 students found technical problems when working with the LAMS built-in mind map tool, which did not correctly save the synchronous modifications (e.g. "*everything worked fine, except for the mind map tool*").

With these findings, it can be concluded that GLUE!, together with the VLEs and the external tools, facilitated the collaboration among students during the enactment of the four representative experiments presented here. It seems reasonable to think that suchlike results would be obtained in other similar collaborative learning situations that require the integration of external tools, and so, it can be concluded that GLUE! meets **REQ2**. Finally, it is noteworthy that external tools integrated through GLUE! can overcome the limitations of built-in ones, such as the LAMS built-in mind map tool. Besides, practitioners can benefit from an even broader variety of tools, thus enriching the individual and collaborative activities that can be enacted.

5.4.3. Integration of existing and popular VLEs and external tools (REQ3)

Five different existing external tools, namely Google Documents, Google Presentations, Dabbleboard, Doodle and You Decide widget were integrated with GLUE! in two of the most popular VLEs, namely Moodle (more than 66,000 installations in 216 counties at the time of writing this dissertation[2]) and LAMS (more than 6,700 members in the LAMS community across

[2] `http://moodle.org/stats`

80 countries in April 2011[3]), and in one well-known platform, MediaWiki, as part of the four experiments presented in this evaluation. Interestingly, tools like Google Documents and Google Presentations are among the most valued by educators, according to the Top 100 Tools for Learning list[4]. Remarkably, at least 69% of the tools listed there meet the mandatory restriction defined in the GLUE! contract for tools, and so, they could eventually be integrated in VLEs through the GLUE! architecture, as discussed in section 3.4.2. Besides, other popular tools, such as Facebook (in its Live Stream version), Doodle or Kaltura are currently available for integration with GLUE! too (see chapter 4).

Therefore, it can be concluded that GLUE! supports the integration of existing and popular VLEs and tools, which could be employed in these and many other similar collaborative learning situations, thus complying with **REQ3**. The few and widespread restrictions imposed to VLE and tool providers are mainly responsible for the compliance to this requirement. But that is not all, just by developing the appropriate adapters, newly developed VLEs might also integrate existing tools, while newly developed tools might also be integrated in existing VLEs, through the GLUE! architecture.

5.4.4. Integration of many external tools (REQ4)

The educators that instantiated the four experiments could choose among up to 17 external tools (in addition to the regular built-in tools), no matter the VLE they were employing. Significantly, this number, *per se*, is even higher that the number of built-in tools included in most VLE distributions, like those currently offered by Moodle (14), Blackboard (16) or Claroline (8). Eight of these tools were W3C widgets deployed in an Apache Wookie server installed in the GSIC-EMIC premises, although any other Apache Wookie server could have been employed instead, just by registering it in the internal tool registry. Here, it is convenient to point out that any other W3C widget could have been integrated without developing any extra integration code, just by deploying the intended widget in the Apache Wookie server. This is a consequence of using GLUE! as a middleware architecture to achieve interoperability with other loosely-coupled approaches, as explained in 3.7.2. Interestingly, the remaining external tools were hosted by third-parties, except for MediaWiki, which was also offered by the GSIC-EMIC.

Educators highlighted the considerable number of newly available external tools, which opened the opportunity to put into practice new individual and collaborative activities. This is supported, for instance, by comments from the interviews with AN educators, who expressed, their satisfaction for "*the opportunity to use many tools that were not available before*". The number of available external tools in GLUE! is expected to keep growing with the development

[3] `http://lamscommunity.org/dotlrn/clubs/educationalcommunity/forums/message-view?message_id=1261220`. Last visited: June 2012.

[4] `http://c4lpt.co.uk/top-tools/top-100-tools-for-learning-2011`. Last visited: June 2012.

of new tool adapters. This is fostered by the few restrictions imposed in the GLUE! contract for tools, which facilitates the eventual implementation of adapters for many other tools (like most of those listed in the Top 100 tools for Learning).

GLUE! supports the integration of many external tools, as compared with the number of available built-in tools typically found in most VLEs, and so, it can be concluded that GLUE! satisfies **REQ4**. These external tools could be employed in a large number of collaborative learning situations like those presented in the experiments supporting this evaluation, or other similar situations. Besides, due to the promotion of a many-to-many integration, every new integrated tool will be available, at least, in Moodle, LAMS and MediaWiki.

5.4.5. Development effort (REQ5)

The four experiments presented in this evaluation required the integration of five external tools in three VLEs. That integration was carried out by three different developers, who independently programmed the code of three VLE adapters (Moodle, LAMS and MediaWiki) and four tool adapters (Google Docs, Dabbleboard, Apache Wookie and Doodle). These developers had different backgrounds and programming skills, and so, there may exist some variations in the new SLOC and time invested in the development of these adapters due to these issues. Nevertheless, it is important to remark that the GLUE! architecture has been designed to facilitate the implementation of new adapters to external developers, who may also have different backgrounds and programming skills, and that these are empirical estimations of the development effort for the particular case of these external developers. Table 5.7 shows the new SLOC and the time invested for the VLE and tool adapters used in the experiments (marked in italics), and for all the remaining adapters that are currently available, indicating who implemented each of them.

The first two adapters developed for the GLUE! architecture (marked in bold) enabled the integration of Google Docs in Moodle, and were implemented from scratch. From then on, the effort required to develop new tool adapters was significantly reduced. For instance, while the tool adapter for Google Docs took 487 lines of code and 83 man-hours, the tool adapter for Dabbleboard only took about a quarter of the new SLOC and about ten times less time. The tool adapter to integrate any web content was very easily implemented, just by cloning the project of other adapters, deleting some code and adding three new lines (the invested time is rounded up). Others, however, took among 4 to 8 man-hours to be developed (e.g. Doodle and Kaltura adapters). The effort to develop new VLE adapters was also lower taking the Moodle adapter as a reference. However, in this case, an important programming burden was unavoidable due to the graphical interoperability between the VLE and the VLE adapter. Besides, less code could be reused, in this case from the Moodle adapter, due to the different programming languages imposed by VLE contracts. It should be note that only those lines that were explicitly developed

Table 5.7: Study of the new SLOC and the time invested. The first VLE and tool adapters are marked in bold. Those adapters used in the experiments are marked in italics.

Adapter (programming language)	*New SLOC*	*Invested time (man-hours)*	*Developer*
VLE adapter for Moodle (PHP)	***1863***	***415***	***developer1***
VLE adapter for LAMS (Java)	*909*	*373*	*developer1*
VLE adapter for MediaWiki (PHP and Javascript)	*1936*	*202*	*developer3*
Tool adapter for Google Docs (Java)	***487***	***83***	***developer1***
Tool adapter for MediaWiki (Java)	83	8	developer2
Tool adapter for Dabbleboard (Java)	*125*	*8*	*developer2*
Tool adapter for Apache Wookie (Java)	*92*	*8*	*developer2*
Tool adapter for Doodle (Java)	*84*	*4*	*developer2*
Tool adapter for web content (Java)	3	1	developer2
Tool adapter for Noteflight (Java)	86	6	developer3
Tool adapter for Kaltura Video (Java)	97	5	developer3
Tool adapter for Facebook Live Stream (Java)	47	4	developer3

for a new adapter are shown in Table 5.7. This consideration is very important in the case of GLUE! because most of the code demanded to integrate a tool in a VLE can be reused from previous integrations, due to the promotion of a many-to-many integration, the definition of an intermediate software layer, and the establishment of well-defined and loosely-coupled contracts between the GLUE! core and VLE and tool adapters. Further details about the calculation of the new SLOC are provided in the Appendix A of this dissertation.

The incremental development effort required to enact the four experiments with GLUE! also decreased progressively, taking AN-2010 as a reference, even though different VLEs were employed. For example, AN-2010 needed 2475 new SLOC and 506 man-hour; SE-2011 demanded 2028 new SLOC and 210 hours; and ICTE-2012 necessitated 993 new SLOC and 377 hours. Interestingly, once the main VLE adapters are developed and only some new tools need to be integrated, the effort is greatly reduced. For example, if after these situations a new one that employs Moodle, some W3C widgets, Doodle and Noteflight had been devised, then the required development effort to provide computational support for such situation would only have been 86 new SLOC and 6 man-hours. Besides, if after instantiating this situation educators decided to move to LAMS, no additional development effort would be needed (see Figure 5.5b), as opposed to the case in which one-to-one (*ad hoc*) integration approaches are employed (Figure 5.5a).

Considering the data presented here, it can be asserted that GLUE! facilitates the integration of external tools in VLEs, reducing the effort required to develop new VLE and tool adapters, to be used in these and in other similar collaborative learning situations. In the particular case of tools this effort has turned out to be low (about 100 new SLOC and 6-8 man-hours), while in the case of VLEs this effort has been found to be higher, although comparable to that in other

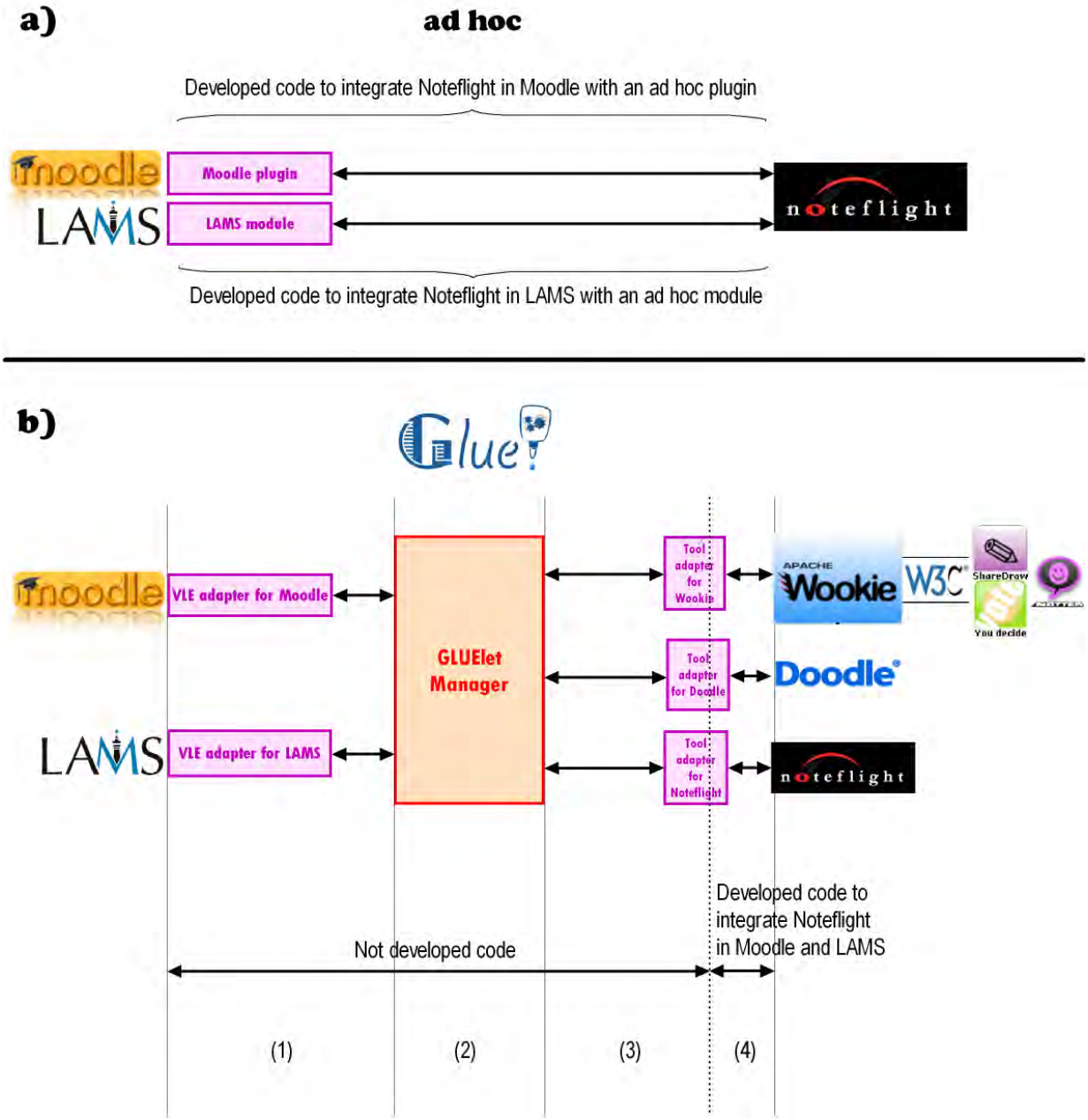

Figure 5.5: Examples of the new SLOC in the integration of Noteflight in Moodle and LAMS: a) without GLUE!; b) with GLUE!. The lower effort demanded with GLUE! is due to: (1) a three-layer architecture that promotes a many-to-many integration, allowing to reuse all the VLE-specific integration code encapsulated in VLE adapters; (2) a three-layer architecture that allocates most of the integration logic in a central element, which is reused for any integration; (3) standardized, simple and loosely-coupled contracts, which allow tool adapters to reuse the code that implements the communication with the GLUElet Manager. Only (4), the part of the tool adapter that implements the tool-specific contract, must be developed.

similar loosely-coupled integration approaches, as it is later discussed in section 5.5.2. Besides, evidences taken from the experiments suggest that the development effort GLUE! requires to enact collaborative learning situations is progressively reduced. For all these reasons, it can be concluded that GLUE! meets **REQ5**.

5.4.6. Built over VLEs and tools (REQ6)

Three VLE adapters (Moodle, LAMS and MediaWiki) and four tool adapters (Google Docs, Dabbleboard, Apache Wookie and Doodle) were employed in the four experiments supporting this evaluation; all these adapters were developed using the interfaces provided by existing VLEs and tools, without modifying a single line of the original code. Besides, other five tool adapters are currently available, being all of them built over existing tools too.

Any VLE administrator can download the appropriate VLE adapter, and install it like any other VLE extension, indicating in a configuration parameter the GLUElet Manager that is expected to receive the requests from this adapter. Any GLUE! administrator can download and install any of the available tool adapters, registering it (and the tools it wraps) in the internal tool registry. Besides, the GLUE! administrator may decide to use some tool adapters provided by third-parties, different from those providing the GLUE! core and the external tools.

After surveying all the available adapters, it is possible to confirm that GLUE! is built over VLEs and tools, rather than modifying their code; thus complying with **REQ6**. The few restrictions on VLEs and tools, and the use of the adapter pattern are mainly responsible for the compliance to this requirement.

5.4.7. Other findings

Some other interesting findings came out from the qualitative and quantitative data obtained from practitioners as part of four experiments. One of these findings concerns the *flexibility of the architecture to support changes in the group settings after the instantiation of the activities*. Both AN educators completely agreed that GLUE! was able to support the reconfigurations they needed in the social distribution of learners. The reorganization of groups and supergroups was the action taken by AN educators in response to some students' absences, which otherwise would have somewhat precluded from drawing the sequence diagram and flowchart in collaboration, as well as from reviewing and getting feedback from the designated supergroup's partners. This assertion was also pointed out in the interview with the ICTE educator, who also needed to reorganize some of the groups of five. Therefore, it can be seen that the update of users sharing external tool instances, which was added to the tool life cycle, turned out to be very useful in real contexts.

Other interesting finding was the perception of a *seamless integration of external tools* by many students. Particularly, ICTE-2012 students, which had never used LAMS before, were asked about whether the tools they employed (both built-in and external) were perceived as integrated in LAMS. Surprisingly, external tools obtained similar results as compared to LAMS built-in tools with, for instance, 82.6% of students in *agreement* or *complete agreement* that Google Presentations was perceived as integrated (see Table 5.8). Qualitative comments also supported this finding: "*All the tools are shown within the platform; I did not feel like moving to another web site at any moment*"; "*All the activities are as a whole in LAMS*"; "*In my opinion, everything is integrated in the platform in a smoothly and very simple way*". Significantly, a seamless integration reduces the learning curve, facilitating that students focus on achieving the learning objectives, rather than on getting used to new software. In GLUE!, this seamless integration is the consequence of transparently managing the retrieval and visualization of external tools instances, keeping the look and feel of both VLEs and tools.

Table 5.8: Answers obtained from the ICTE students to the question about the seamless integration. Statement: "*The x tool is perceived as integrated in the platform employed (LAMS)*", where *x* could be: forum (built-in), mind map (built-in), presentation (external), voting (external).

Options	*forum*	*mind map*	*presentation*	*voting*
Completely agree	5/23 (21.74%)	5/23 (21.74%)	7/23 (30.43%)	4/23 (17.39%)
Agree	**15/23 (65.22%)**	**15/23 (65.22%)**	**12/23 (52.17%)**	**11/23 (47.83%)**
Somewhat agree	3/23 (13.04%)	2/23 (8.70%)	4/23 (17.39%)	5/23 (21.74%)
Somewhat disagree	0/23 (0%)	1/23 (4.35%)	0/23 (0%)	2/23 (8.70%)
Disagree	0/23 (0%)	0/23 (0%)	0/23 (0%)	0/23 (0%)
Completely disagree	0/23 (0%)	0/23 (0%)	0/23 (0%)	0/23 (0%)
No answer	0/23 (0%)	0/23 (0%)	0/23 (0%)	1/23 (4.35%)

Findings regarding limitations in GLUE! also appeared during these experiments. For example, AN educators complained in AN-2011 about having to *instantiate the collaborative learning situation again*, even though the design was exactly the same that in AN-2010: "*I would have liked to reuse some of the instantiation work done in the last year*". This was actually a limitation in Moodle, which unlike other VLEs like LAMS, does not support the design of generic structures of activities to be instantiated in different courses, and eventually, with different students. However, ongoing research aims at proposing a new independent element in the GLUE! core, called GLUE!-PS (GLUE!-Pedagogical Scripting), which could interoperate with the GLUElet Manager to enable the batch deployment of generic learning designs in different VLEs [Mun12b, Pri11]; these designs could be created outside of VLEs and could also include external tools integrated through the GLUE! architecture. This new module could be employed to enable the instantiation of the same design in different courses, and even in different VLEs, thus fostering the reciprocal sharing of learning designs among practitioners.

Another limitation that appeared during the experiments was the *dependence on tool providers*. Those works that, like GLUE!, tackle the integration of external tools in VLEs rely on the fact that tool contracts have been thoughtfully designed, and so, they are not very likely to change (at least without prior notice). Nevertheless, on February 23, 2012, during the ICTE-2012 experiment, Google modified the API of Google Presentations[5] in a way that the retrieval and visualization of created instances through the Google Docs adapter stopped working. This was an important setback that precluded half of the 51 original ICTE-2012 students from enacting this and the subsequent activities. Fortunately, due the loosely-coupling of the architecture and the independent implementation of adapters, *developer 1* fixed this problem in less than one hour, modifying the Google Docs adapter to meet the new Google Presentations contract. The remaining 25 ICTE-2012 students were able to accomplish all the activities designed for this experiment without any problems in their corresponding face-to-face session (February 27), providing useful evaluation data. In any case, it is convenient to be aware of this dependence in those approaches integrating external tools in VLEs. Nevertheless, the alternative of integrating just *ad hoc* developed tools and VLEs, which could overcome this limitation, faces other important shortcomings, as discussed in the introduction of this dissertation, hindering its adoption by real practitioners.

5.5. Comparison with other loosely-coupled integration works

After the analysis of the compliance to the stakeholders' requirements, this section presents a feature analysis and a comparison of the development effort in GLUE! and in other many-to-many approaches with a similar degree of integration [Ala10c]. Here, the objective is to discuss the compromises between the kind of tools that can be integrated, the degree of functionality that is supported and the development effort that is demanded. Tight approaches are out of the scope of this comparison for several reasons. First, current trends in software interoperability are moving towards a low degree of coupling, especially with the rise of web technologies and SaaS applications in the last few years. Besides, although a tight approach may be useful to accomplish the particular integration of one tool in one VLE, the literature shows that those tight approaches intended for the generic integration of different tools in different VLEs have few chances to succeed due to the high development effort demanded. In fact, and as an illustrative example, two *ad hoc* loosely-coupled approaches that offer a low degree of functionality are also included in the comparison of the development effort, just to help readers to get an idea on the amount of effort demanded.

[5] `http://googleappsupdates.blogspot.com.es/2012/02/new-presentations-editor-is-now-default.html`. Last visited: June 2012.

5.5.1. Feature analysis

GLUE!, Apache Wookie and Basic LTI are the three main loosely-coupled approaches that foster the many-to-many integration of external tools in VLEs [Ala10c]. They all enable the communication between VLEs and tools by means of adapters that meet the contracts defined by their corresponding approaches: GLUE! integration contracts (see sections 3.4.2 and 3.4.3), Wookie REST API[6] and IMS Basic LTI specification [IMS10b]. These contracts detail the functionality supported, and so, they can be taken as a reference for the feature analysis performed in this section. Nevertheless, it is important to note that not all the adapters developed for these approaches fully implement these contracts (e.g. the Wookie block for Moodle[7] only implements a part of the Wookie REST API). Table 5.9 analyzes the compliance to the most representative features supported by either of these approaches.

Table 5.9: Feature analysis comparing the three main loosely-coupled integration approaches.

Tag	*Feature*	***GLUE!***	***Apache Wookie***	***Basic LTI***
F1	*Creation of external tool instances*	✔	✔	✘
F2	*Deletion of external tool instances*	✔	✘	✘
F3	*Retrieval of external tool instances*	✔	✔	✔
F4	*Clone of external tool instances*	✘	✔	✘
F5	*Configuration of external tool instances*	✔	✔	✘
F6	*Update of configurations in external tool instances*	✘	✔	✘
F7	*Update of users sharing external tool instances*	✔	✔	✘
F8	*Authorization for the management of external tool instances*	✔	✘	✘

The Wookie server and GLUE! have been designed to allow the creation of multiple instances of external tools (*F1*). This feature can be combined with the VLE functionality to manage different group structures, so that each of these instances may be assigned to one single learner or to a group of learners. As these instances are created, they should also be destroyed, and so, GLUE! features the deletion of tool instances (*F2*). The Apache Wookie contract does not explicitly support the deletion of individual instances; however, when a widget is deleted from a Wookie server all its instances are destroyed. Of course, the instances that are created can be retrieved at any moment with these two approaches, complying with *F3*. Finally, Apache Wookie adds an extra feature (*F4*) to clone external tool instances. This feature can be offered without much difficulty by Apache Wookie, since it integrates simple and homogeneous widgets.

Basic LTI defines the concept of *address*, instead of the concept of instance. Every Basic LTI provider typically exposes one address that allows to access some of the tool functionality.

[6] `https://cwiki.apache.org/WOOKIE/wookie-rest-api.html`. Last visited: June 2012.
[7] `http://moodle.org/mod/data/view.php?rid=3319`. Last visited: June 2012.

Thus, Basic LTI does not support the creation, nor deletion, nor cloning of external tool instances, and all the learners access the same abstraction of the external tool with this approach. That access (called *launch* in Basic LTI) is the only request that can be sent from consumers to providers, and it is the equivalent to the retrieval of external tool instances (*F3*).

Apache Wookie and GLUE! feature the configuration of external tool instances, thus meeting *F5*. This feature allows educators to configure external tool instances within VLEs. Besides, the combination of *F5* with *F1* enables the configuration of instances of the same tool in a different way for each group defined in the VLE. Both integration approaches support different parameters (or properties in the Apache Wookie terminology) for every tool. In the case of GLUE!, the parameters that can be configured are predefined in the XForms or HTML5 templates, while in the case of Apache Wookie these parameters are predefined in the code of each W3C widget. On the contrary, Basic LTI does not allow the particular configuration of a tool. Though some parameters are characterized in the Basic LTI request between the consumer and the provider[8], these are very generic and are only intended for administrative purposes (e.g. information regarding the user, the course, or the organization).

With GLUE!, the configuration values need to be set before the actual creation of tool instances (early-binding), since only one request is sent to the GLUElet Manager for the creation and configuration of each tool instance. This is more efficient in terms of communication, but less flexible, precluding from updating the configuration once the tool instances are created (failing in *F6*); however users can generally retrieve the instances and graphically manipulate their content and appearance. In contrast, Apache Wookie supports a late-binding, first creating and then configuring the instances of W3C widgets (complying with *F6*). That was easy to implement here, because unlike in the case of GLUE!, an homogeneous contract is imposed to the external tools that can be deployed in Apache Wookie.

Both Apache Wookie and GLUE! support the update of users sharing external tool instances (*F7*). Therefore, modifications in the structure of VLE groups can be propagated to the external tool instances that have been previously created, so that these instances are updated to be shared according to the new distribution of VLE users. Basic LTI only retrieves one instance for any user in any activity without taking into account the group settings. Changes in these settings do not have any consequences in the integrated tools, and so *F7* does not apply for Basic LTI.

Some external tools require end-users, or the applications that interact with these tools, to be authorized. The GLUE! architecture provides two generic compromise solutions for this purpose, thus featuring *F8*. The first one is a weaker solution that uses centralized institutional credentials and is transparent to end-users. In the second one, users have to manually solve

[8] `http://simplelti.appspot.com/dotest`. Last visited: June 2012.

the security dependencies imposed by external tools using either OAuth, OpenID or native tool credentials. Unlike GLUE!, Apache Wookie does not need to implement any special software to grant the authorized management of widget instances, since security issues are not considered in the homogeneous contract defined for these widgets. Basic LTI uses the OAuth protocol in the requests from Basic LTI consumers to Basic LTI providers, but just to sign the messages (two-legged OAuth). Therefore, the authorization for the management of external tool instances is not taken into account in this approach.

All in all, it can be concluded that GLUE! offers a higher degree of functionality as compared to IMS Basic LTI, enabling also the integration of heterogeneous external tools, unlike Apache Wookie (which requires all of them to meet the W3C specification). In this point, it is convenient to assess the consequences of offering this higher degree of functionality and supporting the integration of a wider set of external tools in the development effort, since these are all interrelated issues. Remarkably, GLUE! can complement Apache Wookie and Basic LTI rather than compete with them, leveraging these other loosely-coupled approaches the affordances of the GLUE! architecture.

5.5.2. Development effort

Figure 5.6 compares the new SLOC in the three main loosely-coupled integration approaches that also foster a many-to-many integration, namely GLUE!, Basic LTI and Apache Wookie, and in some examples of *ad hoc* (one-to-one) approaches, which integrate Facebook Live Stream[9] and Kaltura[10] in Moodle with a similar degree of functionality and coupling than the GLUE! architecture. The elements of Basic LTI and Apache Wookie that have been selected to illustrate this comparison are: the Basic LTI consumer for Moodle[11], the Basic LTI provider for Apache Wookie[12], the Basic LTI provider for MediaWiki[13], the Basic LTI provider for Noteflight, and the Wookie block for Moodle. Data regarding the new SLOC in these software elements could be obtained without much difficulty, since they are all open source projects, except for the Basic LTI provider for Noteflight. In this particular case, the President of Noteflight LLC, Joseph Berkovitz, which was also the developer of the Basic LTI provider for this tool, was queried about its development effort[14]: "*I actually did the implementation myself, and I would estimate it took about 150 lines of code*". This value was taken as the new SLOC for the Basic LTI provider for Noteflight.

[9] `http://moodle.org/mod/data/view.php?d=13&rid=3316`. Last visited: June 2012.
[10] `http://kaltura.org/moodle-kaltura-plugin`. Last visited: June 2012.
[11] `http://code.google.com/p/basiclti4moodle`. Last visited: June 2012.
[12] `http://code.google.com/p/basiclti4wookie`. Last visited: June 2012.
[13] `http://code.google.com/p/basiclti4mediawiki`. Last visited: June 2012.
[14] Private communication.

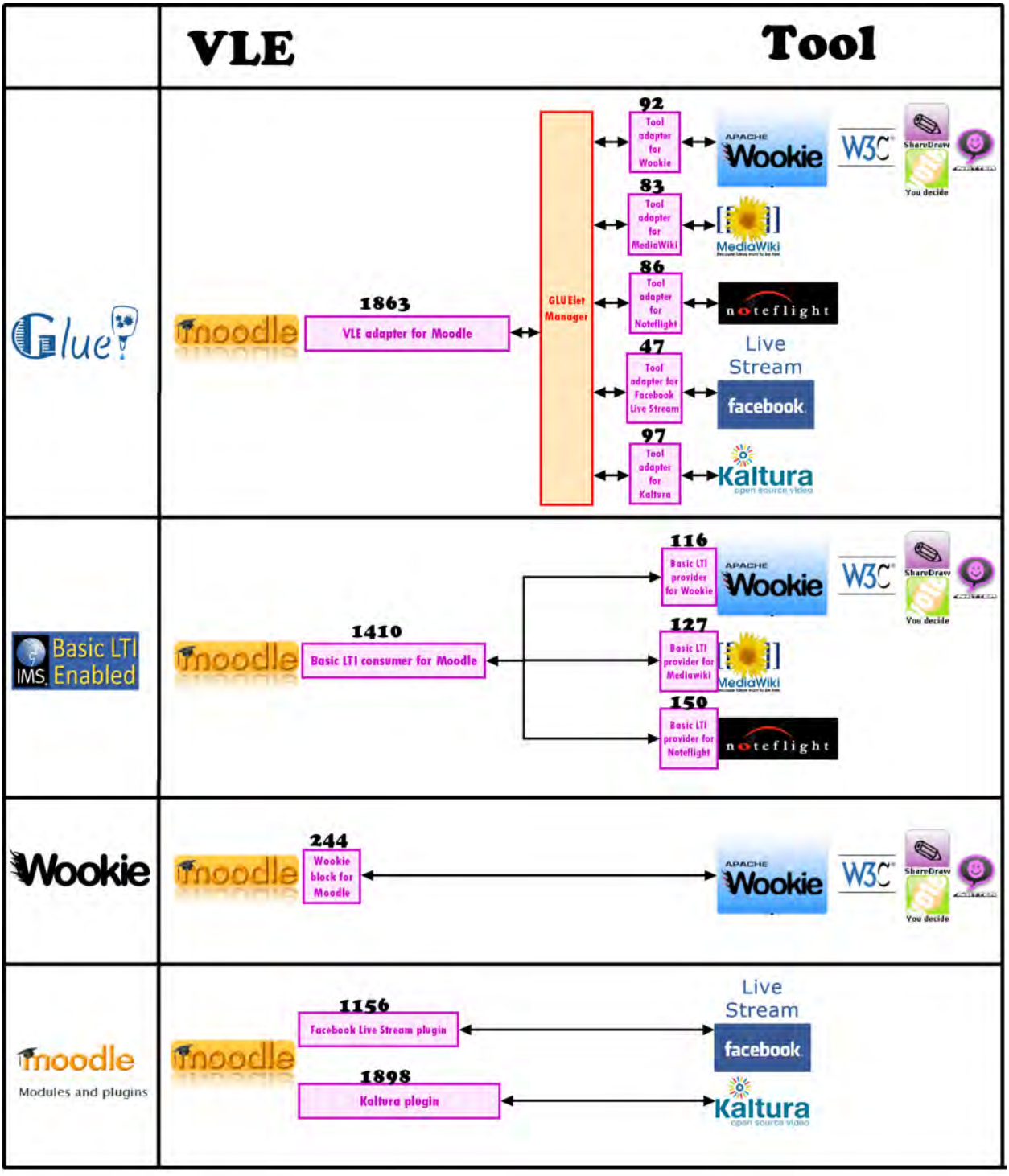

Figure 5.6: Examples of the new SLOC in loosely-coupled integration approaches: GLUE!, Basic LTI, Apache Wookie and Moodle plugins.

Interestingly, the Basic LTI consumers/providers, the Wookie block, the Moodle plugins and the GLUE! VLE adapter for Moodle were developed in PHP. In contrast, the GLUElet Manager and all the GLUE! tool adapters were implemented in Java. This is emphasized here, since PHP projects tend to require fewer lines to implement the same functionality as compared to Java projects [Wal10]. Besides, due to the different features supported by each integration approach and the limitations of this metric (mentioned in section 5.2.2), among which the lack of consideration to developers' skills outstands, only different orders of magnitude should be compared when drawing conclusions about the data shown in Figure 5.6.

First of all, it is convenient to distinguish the distribution of the responsibilities in the examples provided. GLUE! is a three-tier architecture in which the GLUElet Manager partially supports the features analyzed in section 5.5.1, defining also one integration contract that VLE adapters must meet, and another integration contract that tool adapters have to comply with. Those that want to integrate tools in VLEs with GLUE! need to develop either tool adapters or VLE adapters. Basic LTI proposes a two-tier architecture in which the integration contract that consumers and providers must meet is given by the own IMS Basic LTI specification. In the same way that in GLUE!, those integrating tools in VLEs with Basic LTI need to develop software to meet this specification in either the consumer or the provider side. Apache Wookie is also a two-tier approach in which most of the features supported are in the Apache Wookie server, while VLE blocks or modules query this server to make use of these features; those integrating tools in VLEs with Apache Wookie need to develop VLE blocks and, if tools do not meet the W3C Widgets specification, redevelop or adapt them. Finally, *ad hoc* plugins are typically one-tier approaches in which all the functionality to integrate tools in VLEs is implemented in one single software component. Therefore, when comparing the new SLOC in different approaches, it is important to take into account all the elements that must be developed.

For instance, the new SLOC in the GLUE! VLE adapter for Moodle (1863) and in the equivalent for Basic LTI (1410) were of the same order of magnitude, although the latter offers a much more limited functionality, as discussed in section 5.5.1. Interestingly, the development effort in the case of tool adapters was also similar for both approaches, as it can be argued after comparing, for instance, the GLUE! tool adapter for MediaWiki (83) and the corresponding Basic LTI provider (127). That also includes the integration of W3C widgets through GLUE! (92) and Basic LTI (116) in any supported VLE, which otherwise was feasible only for Moodle through a block that does not implement the full Apache Wookie contract, and demanded a few more code lines (244), although in the same order of magnitude.

Once VLE adapters are implemented, trying to develop *ad hoc* plugins typically demands a significantly higher development effort. For example, after the development of the Moodle adapter, Kaltura was integrated in Moodle (and in any other supported VLE) with about 100 new SLOC. However, the equivalent Moodle Module for Kaltura needed one order of magnitude

higher, and only for one VLE (1898). The same comparison can be made between the GLUE! tool adapter for Facebook Live Stream (47) and the Facebook Live Stream plugin (1156). Significantly, the initial effort on a GLUE! adapter for Moodle is worth after integrating two external tools like Kaltura and Facebook Live Stream with *ad hoc* plugins.

All in all, it can be seen that the development effort in GLUE! is similar to that in Basic LTI, although the latter offers much less functionality and features. Besides, that effort is typically much lower than in the case of *ad hoc* approaches with a similar degree of functionality, especially in those cases where multiple tools need to be integrated, or where the VLE intended for these *ad hoc* works is already available with GLUE!. Finally, Apache Wookie seems to demand less effort than GLUE!. Nevertheless, Apache Wookie forces tool providers to develop their tools following the W3C specification. Most valued tools for education do not meet this specification, as previously discussed in this dissertation, and so, trying to integrate popular tools like Google Docs with Apache Wookie would entail a significant extra effort to adapt these tools (if possible) to the W3C Widgets specification.

5.6. Conclusions

The evaluation of the GLUE! architecture was designed to cover the technological and educational dimensions involved in this research. This evaluation was supported by the CSCL-EREM framework and by four authentic experiments that were instantiated and enacted by real practitioners in higher education courses. An evaluation based on authentic experiments was chosen here to show that GLUE! can regularly be employed to support true collaborative learning situations that require the integration of external tools in VLEs. The educational dimension of this research was mainly covered by means of a particularization of the mixed method, which served to collect and analyze the data obtained from these experiments. In contrast, the technological properties were assessed using a feature analysis supplemented with empirical evidences of software complexity.

Evaluation data from these experiments helped to conclude that GLUE! meets the stakeholders' requirements. GLUE! reduced the instantiation time in more than 80% when combined with VLEs like Moodle or LAMS, thus facilitating the instantiation of individual and collaborative activities that require the integration of external tools, as corroborated by the educators involved in the experiments. GLUE! also facilitated the enactment of the collaborative activities, getting, for example, that the visualization of the work done by other group members was easy or very easy for more than 72% of learners. Besides, educators could pick from up to new 17 external tools in the experiments, some of which were among the highest rated for learning purposes. These tools were integrated just by programming about 100 new lines of code and without modifying the implementation given by their providers.

After comparing the GLUE! architecture with other loosely-coupled approaches, it can be concluded that GLUE! offers more functionality and features than Basic LTI, with minor differences regarding the development effort. Apache Wookie, however, initially demands less effort than GLUE!, but it only allows the integration of W3C widgets. Nevertheless, if these widget are intended to be used within one of the VLEs available for GLUE!, it may be worthy to integrate them through the GLUE! architecture. In fact, this was already done, being W3C widgets currently available for Moodle, LAMS and MediaWiki through GLUE!. This reinforces the idea of GLUE! as a middleware architecture for loosely-coupled approaches. Finally, evidences of the development effort indicate that a loosely-coupled *ad hoc* integration is the worst option, especially if the tool is intended to be used within different VLEs.

The evaluation of GLUE! was the fourth and last partial objective of this research. Therefore, after the accomplishment of this evaluation the global objective presented in the introduction of this document is reached. Next chapter summarizes the main conclusions and the challenges in the form of future work related to the proposal presented in this dissertation.

Chapter 6

Conclusions and future work

This chapter harvests the main conclusions of the dissertation, summarizing also the contributions of this research work. It does so in section 6.1, while identifying the main directions of future work in section 6.2, some of which have already been mentioned along this document.

6.1. Conclusions of the dissertation

This dissertation has tackled the limitation regarding the restricted set of built-in tools included in VLEs (e.g. Moodle or LAMS) for the support of collaborative learning situations. This limitation is recurrently reported in the literature, and hinders the instantiation and enactment of many individual and collaborative activities. Pioneering research works tried to overcome this limitation by designing and developing more flexible and tailorable VLEs and tools. Nevertheless, this type of solution has barely been adopted by practitioners, mainly due to the learning effort they needed to master new VLEs or tools, and sometimes because of the impositions of their own institutions, which had already adopted certain VLEs and tools.

The integration of existing external tools in existing VLEs positioned as an alternative to these pioneering works. Nevertheless, those works in the literature tackling such integration are not without problems. One of the most important problems is the high development effort required to integrate new VLEs and tools. This effort, necessary to interoperate the heterogeneous integration contracts defined by VLEs and tools, is significantly high in most approaches due to the promotion of a tight integration, and the adoption of a one-to-one multiplicity (*ad hoc* integration). Another problem worth mentioning is the strict restrictions imposed to VLE and tool providers, which preclude many popular VLEs and tools from being integrated. Finally, some works do not take into account how external tools should be managed when instantiating and enacting collaborative learning situations, thus precluding practitioners from taking advantage of the collaborative features provided by VLEs.

The limitations found in related integration works were further studied in chapter 2 of this document, as part of the analysis of the integration problem particularized for a representative set of popular VLEs and tools, being this the first partial objective of the dissertation. The analysis of the integration problem resulted in the identification of the six main stakeholders' requirements, which were formulated considering the perspectives of three different actors. Practitioners want to instantiate individual and collaborative activities with and attainable effort, and enact them in collaboration with their peers. Also, practitioners want to choose from a wide set of integrated tools when instantiating and enacting these activities, but without having to give up the VLEs and tools they are used to. Developers may be interested in the integration of new tools and VLEs, as long as it does not take much development effort. This integration should be built without modifying the code of existing VLEs and tools, as required by providers.

The analysis of the integration problem also brought the identification and discussion of the main design issues and alternatives that should be considered when proposing new integration approaches. Interestingly, these design issues are interrelated, thus requiring a trade-off on the technical and functional alternatives chosen in order to meet all the stakeholders' requirements. The recommended alternatives suggested in this dissertation to meet all these requirements, overcoming this way the limitations of previous related works in this context, are: to impose few and popular restrictions on VLE and tool providers; to foster a many-to-many and loosely-coupled integration to reduce the development effort; and to offer enough functionality to, at least, facilitate the management of the life cycle of external tools (i.e. creation, configuration, retrieval, deletion and update of external tool instances). The *identification of the main stakeholders' requirements and design issues* is an original contribution of this dissertation, and can be found in [Ala10a, Ala10c, Ala12a].

On this basis, and in order to achieve the second partial objective of this dissertation, a middleware architecture called GLUE!, was proposed in chapter 3. GLUE! is a three-tier loosely-coupled architecture composed by an intermediate element (GLUElet Manager), and two sets of adapters (VLE adapters and tool adapters). GLUE! fosters the many-to-many integration of existing external tools in existing VLEs, imposing few restrictions that main VLE and tool providers currently comply with. Besides, this architecture is designed so that external developers may contribute to extend the integrated VLEs and tools by developing new VLE or tool adapters. Regarding the functionality offered, GLUE! mediates in the management of the tool life cycle, thus facilitating practitioners the instantiation and enactment of both individual and collaborative activities that require the integration of external tools within the VLE interface. The *GLUE! architecture*, including the elements and their responsibilities, the integration contracts, and the restrictions on VLE and tool providers has been published in [Ala10a], being the main original contribution of this dissertation.

The GLUE! architecture has been implemented in the form of a reference implementation, called GLUE!-RI (see chapter 4), as established in the third partial objective of this dissertation. GLUE!-RI is a software distribution that includes a reference implementation of the GLUElet Manager, and a set of examples of VLE and tool adapters. GLUE!-RI demonstrates that the GLUE! architecture is not only a theoretical proposal, but also an actual software system that can be used by practitioners in real educational scenarios. Remarkably, GLUE!-RI is just a reference implementation, and so, other developers may decide to add new elements (e.g. new VLE and tool adapters) or to redevelop the existing ones. Nevertheless, as long as these developers preserve the ideas under the GLUE! proposal (e.g. meeting the integration contracts defined by the architecture), these new elements could interoperate with the ones currently available for GLUE!-RI. The code of the GLUE!-RI elements (available at `http://gsic.uva.es/glue`) can be downloaded and installed by anyone interested in using GLUE! for the integration of external tools in VLEs. Besides, the source code is available under a GPL license to be reused for the development of new elements for the GLUE! architecture. *GLUE!-RI* is the third original contribution of this work, being first mentioned in [Ala12a], and reviewed for the particular cases of Moodle and LAMS in [Ala11b, Ala12c].

The evaluation of this dissertation aimed at showing that GLUE! meets the six stakeholders' requirements, overcoming the limitations of previous related works (see chapter 5). This evaluation was supported by the CSCL-EREM, which is a highly advisable framework for the systematic evaluation of systems and tools that promote collaborative learning. Four authentic experiments involving real university-level educators and students were formalized, and served to collect data and evidences to support this evaluation. The preparation and realization of the four experiments spanned approximately seventeen months, from October 2010 to February 2012, and corresponded to the instantiation and enactment of three different collaborative learning situations in two school years. These situations were designed by educators with different backgrounds on engineering and pedagogy, and met the needs of some of the courses delivered at the University of Valladolid. Apart from different knowledge domains, these situations covered several collaborative strategies, durations, VLEs and external tools, thus making the experiments associated to these situations a set of representative scenarios of real higher education practices.

The results collected from the four authentic experiments served to assert that GLUE! meets the six stakeholders' requirements, what is expected to foster the adoption of this architecture by different actors. These experiments allowed to observe, for instance, that GLUE! can facilitate the instantiation and enactment of individual and collaborative activities, reducing the instantiation time in more than 80% (when combined with VLEs like Moodle or LAMS). Besides, GLUE! can integrate many tools (at least 17 as of this writing) and does not require practitioners to change their commonly used VLEs. These important advantages are expected to motivate practitioners in the adoption of this architecture. Concerning other actors, GLUE! is intended

to encourage the contributions from developers, since it reduces the development effort and facilitates the independent development of adapters, due to its loosely-coupling, its intermediate software layer and its many-to-many integration approach. Significantly, these contributions can be made without modifying the original VLE or tool code, as expected by providers.

In order to complete the evaluation, the features offered by GLUE! and the development effort demanded to integrate new tools and VLEs were compared with those in other loosely-coupled approaches that also foster a many-to-many integration: Apache Wookie and Basic LTI. This comparison led to the conclusion that GLUE! offers more features than Basic LTI, but without significant differences regarding the development effort. This effort is lower in the case of Apache Wookie, but at the cost of enabling only the integration of those tools developed as W3C widgets. In any case, practitioners should not only see GLUE! as a competitor of Apache Wookie and Basic LTI, but also as an alternative to integrate W3C widgets, Basic LTI compliant tools, as well as many other tools that cannot be integrated through Apache Wookie or Basic LTI. Remarkably, GLUE! can interoperate with these two other loosely-coupled integration works because they all share similar technologies, and the functionality offered by Apache Wookie and Basic LTI fits in the life cycle of external tools supported by the GLUE! architecture.

The *evaluation of the GLUE! architecture and GLUE!-RI* has been published in [Ala12a, Ala12b] and completes the last partial objective of this dissertation. Overall, it is possible to state that *this dissertation has designed, developed and evaluated a middleware architecture, showing that it enables the integration of multiple existing external tools in multiple existing VLEs, demanding an attainable development effort to integrate new VLEs and tools, imposing only basic restrictions that most VLE and tool providers already meet, and supporting enough functionality to facilitate the instantiation and enactment of collaborative learning situations.* Therefore, the global objective intended for this dissertation has been achieved.

Finally, and although the GLUE! architecture is intended for the particular context of integrating tools in VLEs, the ideas underlying the proposal of the GLUE! architecture can be relevant to address the problem of integrating external applications to support collaboration in other platforms, such as wikis, social networking sites or PLEs. Actually, a VLE adapter for MediaWiki is available, thus enabling the integration of external tools within this platform, even though MediaWiki was not designed for the educational domain, and lacks important VLE features, like the management of groups and activities. This is an interesting emergent property of this research that was not included among the expected contributions, but that deserves a special mention.

6.2. Future work

During the realization of this work some new issues emerged, being listed for future reviews of this proposal. Besides, some other issues were out of the main scope of this research, and so they were marked with a lower priority. This section suggests a number of lines that could extend and improve the research work presented here, and that include work on these issues. Some of these lines have been mentioned throughout this document, and a few of them are already being addressed in ongoing research. For better understanding these lines, they are classified in four categories: integration of new systems, improvement of integrated systems; integration with other systems in the life cycle of collaborative learning situations; and transfer programs.

The main contribution of this dissertation, the GLUE! architecture, enables the integration of external tools in VLEs. Three VLEs and at least 17 external tools (any W3C widget integrated through Apache Wookie could be used through GLUE! too) are currently available for practitioners. These numbers are expected to keep growing with the *integration of new systems*:

- **Integration of new learning platforms**. Other VLEs are planned to be interoperable with GLUE! thanks to the development of new VLE adapters. These are the cases of Sharepoint LMS (for which some efforts have been put, as part of a Spanish National research project in partnership with the *élogos* company) and Sakai. Besides, there are plans for the migration to new VLE versions, in order to keep the set of VLE adapters updated. This migration is prioritized in the case of Moodle, for which the 2.2 version is already available (being the corresponding VLE adapter working for the 1.9 version). It is noteworthy that the development effort estimated for this migration is low, since most of the integration code can be reused. Apart from VLEs, other learning platforms for which the development of a VLE adapter is under consideration are: the CMS Drupal, and the PLE Southampton Learning Environment. However, in these platforms there is further research to make due to the conceptual differences as compared to VLEs, regarding, for instance, the management of roles, groups and activities.
- **Integration of new tools**. Some other tools, e.g. the PiratePad web-based collaborative text editor, are planned for the GLUE!-mediated integration in VLEs through the development of new tool adapters. Nevertheless, these adapters are expected to be developed without demanding much time and effort, and so, they will be prioritized upon request of those educators using GLUE! for the integration of external tools. Apart from tool adapters developed for the integration of concrete tools, there are also plans for the implementation of a tool adapter acting as a Basic LTI consumer, in order to integrate Basic LTI compliant tools in VLEs through the GLUE! architecture. This corresponds to the idea of GLUE! seen as a middleware architecture of loosely-coupled integration approaches, in the same way that now happens with Apache Wookie.

- **External developments**. All the available VLE and tool adapters were implemented by developers belonging to the research group in which this dissertation has been carried out. Nevertheless, external developers are expected to contribute to the integration of new VLEs and tools, due to the attainable effort demanded and the many-to-many integration fostered. Regarding VLEs, contacts were maintained with *élogos*, a distance learning private company, for the development of the Sharepoint LMS VLE adapter. In the tool side, two tool providers have shown interest in developing tool adapters: the University of Patras (to integrate the Synergo mapping environment); and the University of Sydney (to integrate the EVA3 video learning environment).

Future work also includes some issues that were out of the scope of this dissertation, and thus need to be more thoroughly studied. The clearest examples are the security issues, which were succinctly studied in this document, showing that they can be tackled, although they may need a further analysis and more elaborated solutions. Besides, there are several implementation tasks that could not be completed as of this writing due to time restrictions, and so, they need to be reorganized and prioritized. All these lines of work are classified as the *improvement of integrated systems*:

- **Review the proposals addresing the user level authorization for the management of external tools**. Two compromise solutions advised for different contexts were proposed to address this particular security issue: a weak solution in which institutional credentials are centralized in the internal tool registry; and a stronger distributed solution in which educators need to explicitly grant access to data in external tools to tool adapters. Nonetheless, the first solution is limited by tool providers that may disagree with the policy of sharing institutional credentials among a large number of users. The second solution, however, may cause errors in those cases where tokens or credentials expire and the requests in which these tokens or credentials should be renewed do not begin with actions of the educator in the VLE user interface. An example is the update of users sharing instances; although the educator modifies the groups in the VLE, the requests for the actual update of users are not launched until one of the users try to access the tool instance for the first time after the group changes (according to the implementation of current VLE adapters). If tokens or credentials are expired, students may need to contact to the educator so that he updates the instances (and also renews the tokens or credentials). In order to overcome this limitation, one of the most appealing solutions would consist on the definition of a security management console, where educators could set, update or remove tokens and credentials; but also where they could see the actions pending due to expired tokens or credentials.

- **Implementation of the remaining security issues**. Apart from the user level authorization for the management of tool instances, the analysis of the security problem helped to detect five more security issues in the GLUE! architecture. The user level authorization for the use of VLEs, and the user level authorization for the access to external tool instances exclusively concerns VLEs and tools (in the given order). Nevertheless, there are three more issues whose particular solutions were outlined, but still need to be concreted and implemented. These are the cases of the VLE level authorization for the use of the GLUElet Manager and the GLUElet Manager level authorization for the use of tool adapters; both can be addressed by respectively registering the VLEs in the GLUElet Manager and the GLUElet Manager in the tool adapters, receiving in both cases a different token that should be included in every request. Finally, the privacy and integrity along the communication channels could be solved by signing all the requests using a public/private key algorithm.

- **Migration to OAuth 2.0**. The stronger distributed solution proposed to address the user level authorization for the management of tool instances supports the OAuth delegated authorization mechanism to enable tool adapters the access to data in external tools, on educators' behalf. The current implementation of the GLUE! security component (GSC) and the available tool adapters are compatible with OAuth 1.0. However, OAuth 2.0 has recently been released as an Internet-draft, and some providers like Google are currently compatible with both versions; others like Twitter still work only with OAuth 1.0. New versions of the GSC and tool adapters intended for OAuth compliant tools should thus support OAuth 2.0 too. No changes should be made in the GLUElet Manager, nor in the GLUE! contracts, due to this migration.

- **Administration client for the internal tool registry**. The internal tool registry stores the information of the available external tools and tool adapters, being queried by the GLUElet Manager. The persisted information must be manually set by the GLUE! administrator when installing and configuring the GLUElet Manager. Some scripts are provided with the reference implementation of the GLUElet Manager to help the GLUE! administrator in this burdensome task. An administration web client is planned to be developed to help GLUE! administrators to populate the internal tool registry, and to update it with new tools.

- **Adoption of HTML5**. According to the GLUE! contracts, the configuration templates provided by tool adapters can follow either the XForms or the HTML5 data format. Nevertheless, the available VLE adapters can only process those templates that comply with the XForms specification. Future improvements in the Moodle, LAMS and MediaWiki VLE adapters are expected to include the actual support to HTML5.

- **Support of Moodle backups**. The Moodle VLE adapter does not meet the Moodle contract regarding the generation of course backups. Thus, at this time, those activities supported by external tools are excluded from these backups, precluding educators from restoring or cloning those Moodle courses in which external tool instances need to be employed. This is an implementation issue that only affects the Moodle adapter, and that is listed for future improvements in this particular adapter.

- **Reuse of instances within LAMS**. The LAMS VLE adapter, unlike the Moodle and MediaWiki adapters, does not currently support the reuse of external tool instances in different activities. Therefore, this is a pending implementation task according to which, the LAMS adapter should be able to offer educators the opportunity to reuse instances from previous activities (in the same LAMS lesson). Interestingly, in LAMS, instances are not created until the deployment of the lesson in the monitoring environment, and so, this adapter should manage a set of references to the instances that are to be created.

It is noteworthy that this dissertation contributes to the instantiation and enactment phases of the life cycle of collaborative learning situations, which was presented in section 2.2.1. Nevertheless, this life cycle includes two other phases, design and evaluation, in which practitioners and other different actors may set additional requirements. Some of the future lines should include the *integration of the GLUE! architecture with other systems in the life cycle of collaborative learning situations* intended to meet these requirements:

- **Deployment of learning designs**. VLEs do not generally allow educators to create abstract non-particularized learning designs to be instantiated in different courses or lessons. On the contrary, authoring tools (e.g. Collage [Her06b]) and learning design languages (e.g. IMS LD [IMS03]) are explicitly proposed to facilitate educators and instructional designers the creation and management of abstract learning designs, especially in collaborative contexts where non-trivial structures of groups and activities need to be defined. These abstract learning designs can be shared among practitioners, and also reused in different contexts. Unfortunately, major VLEs like Moodle, Sakai or Blackboard, cannot interoperate with the most popular authoring tools and learning design languages. LAMS is an exception to this, providing its own integrated environment to design and deploy learning designs, although these designs cannot generally be deployed in other VLEs. Ongoing research on this issue, which affects the design and instantiation phases, aims at proposing a new element in the GLUE! core, called GLUE!-PS (GLUE!-Pedagogical Scripting) [Mun12b, Pri11], to enable the deployment of abstract learning designs generated with multiple authoring tools and learning design languages into multiple VLEs. GLUE!-PS follows the many-to-many and low coupling principles that stem from the research done in this dissertation, and employs similar technologies (e.g. REST services).

Besides, GLUE!-PS has been designed to meet the GLUE! integration contract for VLEs. Therefore, GLUE!-PS can request the creation and management of external tool instances to the GLUElet Manager, in order to particularize the learning designs with external tools before deploying them in VLEs. Using the GLUE! - GLUE!-PS partnership educators may decide to instantiate within the GLUE!-PS graphical interface before deploying in VLEs, or to instantiate directly within the VLE interface. Significantly, no changes were done in any of the GLUE! elements, nor in the GLUE! contracts, to be used by GLUE!-PS. The first experiments with educators using the GLUE! - GLUE!-PS partnership have already been made with results hinting that educators appreciate the value in this partnership [Ala12d].

- **Semantic search of external tools**. The internal tool registry persists information about the available external tools. Part of this information (e.g. the tool name, the description, the provider and the tasks supported) is retrieved by the GLUElet Manager, being shown to educators within the VLE interface when they want to add a new tool to a learning activity. Nevertheless, as the number of available tool increases, educators demand some kind of search filter to select the appropriate tools they are looking for (e.g. indicating the intended tasks). Moreover, the information shown to educators might be extended by the GLUE! administrator in order to provide more accurate details (e.g. the last version of the tool, system requirements, opinions of other peers, etc.). Therefore, the GLUE! administrator, who is in charge of the burdensome creation and update of all this information, also suffers the problem of a large number of integrated tools. There exist a research work aimed at overcoming these two problems with different proposals related to semantics. First, this work leverages the advantages of the semantic search of tools to facilitate educators the selection of external tools within the VLE interface. This, however, would require to enrich the GLUE! contract for VLEs (in order to support these semantic searches), and also to modify the existing VLE adapters. Additionally, this work proposes a linked data [Biz09] infrastructure, called SEEK-AT-WD[1] (Support for Educational External Knowledge About Tools in the Web of Data), for the automatic retrieval of information about educational tools [Rui12a,Rui12b]. This infrastructure could be useful to populate and maintain the internal tool registry. The adoption of SEEK-AT-WD would only require to modify the information stored in the internal tool registry so that this information meets the four main linked data principles [Ber06], although no further changes would be needed in the GLUElet Manager, nor in the GLUE! contracts.

- **Monitoring collaboration among students**. The GLUE! architecture does not provide educators with information about interaction analysis (i.e. how students collaborate throughout the technology-mediated learning activities) [Sol05]. Interaction analysis is not supported by GLUE! due to the trade-off between the development effort demanded

[1] `http://gsic.uva.es/seek`. Last visited: June 2012.

and the functionality that can be offered. VLEs do not provide much data about how students collaborate either, although some tools may allow to inspect, for instance, the conversation of a group of students in a forum thread, or a summary with the changes each participant introduces in a shared document. This information could be arranged and formatted using figures, and might be used by educators when evaluating students' performance. The proposal of an extension of the GLUE! architecture, called GLUE!-CAS (Collaboration Analysis Support for GLUE!) [Rod11, Rod12], aimed at gathering information about the collaboration among students during the enactment phase, is currently under research. GLUE!-CAS also relies on the many-to-many and low coupling principles of GLUE!, harvesting information from multiple data sources (e.g. logs from the GLUElet Manager, history of changes in the content of external tool instances, events in VLEs). Nevertheless, this approach would require some changes in the GLUElet Manager and would impose new requirements on VLE and tool adapters. For instance, in every creation or update request the GLUElet Manager should register in GLUE!-CAS the context for the analysis (indicating the VLE users, their associated instances, and the corresponding external tools). Besides, VLE and tool adapters should be able to reply to periodic queries from GLUE!-CAS reporting events in the VLEs and changes in the content of instances.

- **Integration of geolocated tools**. Some of the current research lines on CSCL are analyzing the challenges involved in the instantiation and enactment of collaborative activities in both the physical and virtual worlds [Iba11]. An ongoing research is studying how the GLUE! architecture could be enhanced by means of augmented reality [Dun09] and geopositioning [Rob11], so that students may interact with the physical world by means of the integration of geolocated tools [Mun12a]. In geolocated tools, a geoposition (latitude and longitude coordinates) would be associated to each external tool instance (in the same way that other configuration parameters). Students using a mobile device should approach these coordinates, and once they reach the intended location they would use an augmented reality browser (e.g. Junaio[2]) installed in their smartphones or tablets to access their tool instances. This scenario could be useful, for example, in an activity in which students must write a collaborative document about a given monument *in situ*. There is an exploratory prototype on this line working with the GLUE! - GLUE!-PS partnership [Mun12a]. However, the associations between external tool instances and geopositions are defined in GLUE!-PS, which is also in charge of deploying these instances in Junaio. Therefore, this prototype did not required any modifications in the GLUE! elements, nor in the GLUE! contracts. Nevertheless, this approach does not work when the educator designs and instantiates directly on the VLE user interface; that would require more research, and probably some changes in the GLUE! architecture.

[2] `http://junaio.com`. Last visited: June 2012.

Finally, it is noteworthy that this is a very applied research work, which is intended to enhance educational practices, especially in the CSCL field. Therefore, a very important future line is the definition of a **transfer program**, in which the contributions generated in this dissertation (in the form of a middleware architecture that enables the integration of external tools in VLEs) are adopted by real practitioners and institutions. The first step in this direction has already been made: the implementation of a usable and useful reference implementation of the architecture that anybody can download and install from the GLUE! website (`http://gsic.uva.es/glue`). Besides, user manuals and videos have been created to explain the innovations introduced by the architecture in the normal operation of the VLEs supported. Finally, negotiations have been conducted for the adoption of this proposal in several institutions, such as the University of Valladolid, the Institute of Applied Ophtalmobiology (associated to the University of Valladolid), and the University of Mondragón.

Appendix A: Study of the development effort

The study of the development effort is supported by two quantitative metrics of software complexity, the *new source lines of code* (new SLOC) developed for an adapter and the *time invested* by a developer in the programming of the code. Measurements of these two metrics were obtained employing the source files of the GLUE!-RI 0.8 version. Besides, measurements of the new SLOC for those software elements following other integration approaches were taken using the code that can be downloaded from the references footnoted in chapter 5. The Basic LTI (Tool) provider for Noteflight was an exception to this; data regarding the new SLOC in this example was directly given by its developer, Joseph Berkovitz.

The *time invested* in the development of VLE and tool adapters was calculated by the own developers, adding up the hours listed in their SCRUM backlogs devoted to programming tasks. That excludes, for instance, the time needed to study and understand the VLE or tool contracts, or the time to generate the documentation of the developed code. Measurements obtained from this metric aim at providing empirical data that may be particularly interesting for those VLE and tool providers that want to implement a fully functional version of an adapter, estimating *a priori* how much time it might take.

The *new SLOC* was calculated using the GeroneSoft Code Counter Pro v.1.32[3], which allows to count the lines in source files implemented in languages like Java, PHP or C/C++ among many others. Therefore, the individual packages containing the source code of VLE and tool adapters were scanned using this program, obtaining two different measures: the *total number of lines in the package* (including comments, headers and blank lines) and the *source lines of code* (without comments, headers or blank lines). After that, the code of VLE and tool adapters was carefully checked, discarding those lines that were completely or mostly reused (e.g. lines where the only changes were attributes or parameters renamed/refactored), thus obtaining the *new SLOC*. Interestingly, despite the fact that comments, headers and blank lines are sometimes considered in the SLOC, they were excluded from this calculation, since the

[3] `http://www.geronesoft.com`. Last visited: June 2012.

purpose of this study is to offer empirical data about how many lines a developer should program to actually accomplish the implementation of an adapter. Table 1 replicates the same table presented in chapter 5 when analyzing the development effort, indicating the new SLOC, but also the source lines of code, and the total number of lines in the different adapters.

Table 1: Study of the new SLOC. First VLE and tool adapters are marked in bold.

Adapter (programming language)	*new SLOC*	*SLOC*	*total lines*
VLE adapter for Moodle (PHP)	**1863**	**2046**	**3870**
VLE adapter for LAMS (Java)	909	10205	15459
VLE adapter for MediaWiki (PHP and Javascript)	1936	8947	12737
Tool adapter for Google Docs (Java)	**487**	**487**	**953**
Tool adapter for MediaWiki (Java)	83	313	598
Tool adapter for Dabbleboard (Java)	125	336	785
Tool adapter for Apache Wookie (Java)	92	335	592
Tool adapter for Doodle (Java)	84	289	542
Tool adapter for web content (Java)	3	159	353
Tool adapter for Noteflight (Java)	86	348	981
Tool adapter for Kaltura Video (Java)	97	400	804
Tool adapter for Facebook Live Stream (Java)	47	405	827

Although the Moodle adapter was taken as a reference for the creation of the other VLE adapters, it was not completely implemented from scratch. This was due to the fact that Moodle providers offer a base template with the structure and some of the code for the development of new activity modules[4], and so, the Moodle adapter was implemented using this template. Figure 1 shows the *total* lines (3870) in the Moodle adapter and the SLOC (2046), which includes the *source* lines, plus those tagged as *both* (i.e. *source* + *comment*). Significantly, almost 98% of the code is in PHP files, although there are a few lines in an XML file.

In the cases of the VLE adapters for LAMS and MediaWiki, developers also started from existing code given by the VLE providers. In LAMS, the forum tool served as the basis for the development of the LAMS adapter, while in MediaWiki the *Editforms* extension[5] was employed. In LAMS, most of the new code was developed in Java files and in Java Server Pages (JSP), being about 16% of the new SLOC in XML files. In MediaWiki, however, most of the new code was in PHP, being about the 25% of the new SLOC in JavaScript files. It is important to note that quite of the code developed to establish the communication between the Moodle adapter and the GLUE! core could be reused in the LAMS and MediaWiki adapters, specially in the latter which also employs PHP. Unlike in Moodle, the new code in the LAMS and MediaWiki adapters is distributed among an important number of files developed with different programming languages,

[4] `http://docs.moodle.org/dev/Modules`. Last visited: June 2012.

[5] `http://mediawiki.org/wiki/Extension:EditPageMultipleInputTextAreas`. Last visited: June 2012.

Filename:	Source:	Comment:	Both:	Blank:	Total:
E:\Moodle\MoodleAdapter\gluelet\configuration_form.php	287	190	28	118	623
E:\Moodle\MoodleAdapter\gluelet\edit.php	208	68	16	69	361
E:\Moodle\MoodleAdapter\gluelet\event_handlers_lib.php	229	86	14	76	405
E:\Moodle\MoodleAdapter\gluelet\index.php	49	23	0	33	105
E:\Moodle\MoodleAdapter\gluelet\lib.php	530	514	48	239	1331
E:\Moodle\MoodleAdapter\gluelet\mod_form.php	54	65	12	38	169
E:\Moodle\MoodleAdapter\gluelet\settings.php	10	13	4	9	36
E:\Moodle\MoodleAdapter\gluelet\version.php	2	9	2	3	16
E:\Moodle\MoodleAdapter\gluelet\view.php	116	36	11	46	209
E:\Moodle\MoodleAdapter\gluelet\db\events.php	109	12	0	30	151
E:\Moodle\MoodleAdapter\gluelet\db\upgrade.php	59	45	3	45	152
E:\Moodle\MoodleAdapter\gluelet\lang\en_utf8\gluelet.php	68	6	1	11	86
E:\Moodle\MoodleAdapter\gluelet\lang\es_es_utf8\gluelet.php	68	6	1	12	87
E:\Moodle\MoodleAdapter\gluelet\lang\es_utf8\gluelet.php	68	6	1	11	86
E:\Moodle\MoodleAdapter\gluelet\db\install.xml	47	4	1	1	53
[Totals] (# of files = 15):	**1904**	**1083**	**142**	**741**	**3870**

Figure 1: GeroneSoft Code Counter Pro screenshot showing the SLOC (*source* + *both*) and the total lines (*total*) for the Moodle adapter.

and so, the values that appear in Table 1 for these two adapters are a rather reliable estimation of the new SLOC, but it cannot be assured that these are accurate values.

Tool adapters were developed taking the Google Docs adapter as a reference, which was completely implemented from scratch. All the new code developed for these adapters is in Java files, as it is exemplified with the Dabbleboard adapter in Figure 2. The package containing this adapter has 785 total lines (336 excluding comments, headers and blank lines), although only 125 of them needed to be programmed when developing this adapter. Significantly, the code of the external libraries used by the adapters (e.g. *GLUEcommon* or RESTlet) is not included here, since these libraries are not packaged together with the individual adapters.

Filename:	Source:	Comment:	Both:	Blank:	Total:
E:\Dabbleboard\Dabbleboard\src\glue\adaptors\implementation\dabbleboard\entities\Dabbleboard.java	207	239	8	86	540
E:\Dabbleboard\Dabbleboard\src\glue\adaptors\implementation\dabbleboard\entities\DabbleboardFactory.java	17	0	0	7	24
E:\Dabbleboard\Dabbleboard\src\glue\adaptors\implementation\dabbleboard\manager\DabbleboardAdaptorApplication.java	15	19	0	6	40
E:\Dabbleboard\Dabbleboard\src\glue\adaptors\implementation\dabbleboard\manager\DabbleboardAdaptorServerMain.java	29	22	1	20	72
E:\Dabbleboard\Dabbleboard\src\glue\adaptors\implementation\dabbleboard\resources\InstanceFactoryResource.java	26	11	0	5	42
E:\Dabbleboard\Dabbleboard\src\glue\adaptors\implementation\dabbleboard\resources\InstanceResource.java	33	24	0	10	67
[Totals] (# of files = 6):	**327**	**315**	**9**	**134**	**785**

Figure 2: GeroneSoft Code Counter Pro screenshot showing the SLOC (*source* + *both*) and the total lines (*total*) for the Dabbleboard adapter.

Appendix B: GLUE! data format

The GLUE! integration contracts for VLEs and tools report that the data exchanged among the elements of the architecture must be formatted following an Atom extension, namely the GLUE! data format. This Atom extension adds some minor elements needed by the architecture (e.g. the tool provider or the tool type). This appendix describes and exemplifies the GLUE! data format in different requests and responses among the GLUE! elements.

B.1 Data format overview

The GLUE! data format is a specialization of the Atom data format, built using the Atom extension mechanism. The GLUE! data format can be applied to:

- HTTP entities returned to a method request that are not just an HTTP status code.
- Call parameters sent in a `POST` request.

Significantly, call parameters in other methods like `GET` or `DELETE` are not formatted because the HTTP entity in these methods must be empty. In these cases, call parameters are serialized in the URL addressed by the method.

As a general rule, the data required by GLUE! follow the semantic established by the Atom protocol. Nevertheless there are some cases in which new XML elements must be defined. These elements use the specific namespace `http://gsic.uva.es/glue/1.0`, as it is defined by the Atom extension mechanism. Significantly, the Atom specification forces the inclusion of some specific elements, typically the title and the last update date, in order to validate an Atom document. This sometimes causes the redundancy of the data included in such elements (and in other elements), due to the lack of more appropriate information to be transmitted.

B.2 Methods of the GLUE! contract for tool adapters

`GET /configuration` (response)

The response a tool adapter returns to a `GET /configuration` call consists of a single `atom:entry` element, containing a configuration template, which follows the XForms or the HTML5 format. The `atom:entry` descendant elements are detailed in Table 2. Besides, an example of response sent by the Google Docs adapter can be seen next:

Table 2: `atom:entry` assignments for `GET /configuration` responses to the GLUElet Manager.

Element	*Value that must be assigned*
`atom:id`	URI of the `/configuration` resource in the tool adapter.
`atom:content`	**Configuration template. This is represented as an `html:div` element containing an XHTML structure that embeds XForms or HTML5 elements.**
`atom:author`	Information about the tool adapter or about its developer. GLUE! built-in adapters include an `atom:author` with the string "Group of Intelligent and Cooperative Systems (GSIC)".
`atom:title`	Textual descriptor that briefly describes the configuration, indicating, for instance, whether it is a default configuration or not. Each tool adapter defines the value of this element.
`atom:updated`	Date of the last modification of the configuration definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom"
    xmlns:glue="http://gsic.uva.es/glue/1.0">
    <author>
        <name>Group of Intelligent and Cooperative Systems (GSIC)</name>
    </author>
    <id>http://localhost:8186/ToolAdaptor/GData/configuration?tool=Google+Documents</id>
    <title type="text">Default configuration definition</title>
    <updated>2010-05-12T06:55:39.26Z</updated>
    <content type="xhtml">
    <div
        xmlns="http://www.w3.org/1999/xhtml"
        xmlns:ev="http://www.w3.org/2001/xml-events"
        xmlns:xf="http://www.w3.org/2002/xforms"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:ins="http://gsic.uva.es/glue/adaptors/implementation/gdata/1.0">
        <!-- model -->
        <xf:model>
            <xf:instance id="input-file">
                <ins:data>
                <ins:file xsi:type="xs:hexBinary"/>
                <ins:title xsi:type="xs:string">Default title</ins:title>
                </ins:data>
            </xf:instance>
```

```
            <xf:submission action="http://inapropiate.url.com/send" id="s"
                method="post"/>
        </xf:model>
        <!-- user interface control in body; only 'ref' bindings are used -->
        <p>
            <xf:input ref="/ins:data/ins:title">
                <xf:label>Title:</xf:label>
            </xf:input>
        </p>
        <p>
            <xf:upload ref="/ins:data/ins:file">
                <xf:label>Select file:</xf:label>
                <xf:filename ref="@filename"/>
                <xf:mediatype ref="@mediatype"/>
            </xf:upload>
        </p>
        <p>
            <xf:submit submission="s"><xf:label>Submit</xf:label></xf:submit>
        </p>
    </div>
    </content>
</entry>
```

This response can be employed in the context of the creation of an external tool instance (see section 3.5.1). More specifically, it corresponds to the interaction 2.4 in Figure 3.2.

`POST /instance` (request)

The HTTP entity that is sent in the request from the GLUElet Manager to tool adapters for the creation of instances (`POST /instance`) must contain an `atom:entry` element, that includes, as the most relevant fields, the configuration for the particular instance and the list of users that are intended to access that instance. Table 3 shows the descendant elements of this `atom:entry`. Besides, an example of this request for the Google Docs adapter is shown next:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<entry xmlns="http://www.w3.org/2005/Atom"
    xmlns:glue="http://gsic.uva.es/glue/1.0">
    <glue:toolName>Google Documents</glue:toolName>
    <glue:configuration>
        <ins:data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:ins="http://gsic.uva.es/glue/adaptors/implementation/gdata/1.0">
            <ins:file filename="test.odt" xsi:type="xs:hexBinary">
                504(...)3600000000
            </ins:file>
            <ins:title xsi:type="xs:string">My first document</title>
        </ins:data>
    </glue:configuration>
    <glue:users>
```

Table 3: `atom:entry` assignments for `POST /instance` requests from the GLUElet Manager.

Element	*Value that must be assigned*
`glue:toolName`	**Parameter "tool" indicating the tool name as a human-readable string.**
`glue:configuration`	**Parameter "configuration" as an XML composed-element. Its content includes the configuration values provided by the educator, once validated.**
`glue:parameters`	**Text string with other "parameters" that the tool adapter may need. Its value comes from the internal tool registry, and must be set when registering a new tool.**
`glue:users`	**Parameter "users" as a list of `glue:user` elements. Each of these elements contains a textual identifier of a user (VLE username) that shares the external tool instance, once created.**
`glue:callerUser`	**Parameter "callerUser" containing a textual identifier (VLE username) of the user that requests the creation of the tool instance.**

```
        <glue:user>
            student1
        </glue:user>
        <glue:user>
            student2
        </glue:user>
        <glue:user>
            student3
        </glue:user>
        <glue:user>
            student4
        </glue:user>
    </glue:users>
    <glue:callerUser>
        teacher
    </glue:callerUser>
    <glue:parameters>feedURL=http%3A%2F%2Fdocs.google.com%2Ffeeds%2Fdocuments%2Fprivate%2Ffull%2F
    </glue:parameters>
</entry>
```

This request can be employed in the context of the creation of an external tool instance (see section 3.5.1). It corresponds to the interaction 3.3 in Figure 3.2.

`POST /instance` (response)

The response from tool adapters to a `POST /instance` consists on an `atom:entry` element with information of the tool instance that is created, being the most relevant item the URL that identifies the instance. Table 4 shows the descendants elements of this `atom:entry`. An example

of this response returned by the Google Docs adapter after the successful creation of a Google Documents instance is shown next:

Table 4: `atom:entry` assignments for `POST /instance` responses to the GLUElet Manager.

Element	*Value that must be assigned*
`atom:id`	URI of the `/instance/{instanceId}` resource that has been created in the tool adapter.
`atom:link, with rel="alternate"`	**URL of the real resource that has been created. It must be accessible from a web browser.**
`atom:author`	Information about the tool adapter or about its developer. GLUE! built-in adapters include an `atom:author/atom:name` element with the string "Group of Intelligent and Cooperative Systems (GSIC)".
`atom:title`	Textual descriptor of the instance that is created. Its representation depends on the specific tool or tool adapter.
`atom:updated`	Date of the creation of the instance.

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom"
    xmlns:glue="http://gsic.uva.es/glue/1.0">
    <author>
        <name>Group of Intelligent and Cooperative Systems (GSIC)</name>
    </author>
    <id>http://localhost:8186/ToolAdapter/GData/instance/1</id>
    <link
        href="http://docs.google.com/Doc?docid=0AUMiWAp0au_cZGdmend4cl8yNjFjNjcydGdmaA&amp;hl=en"
        rel="alternate" title="HTML accesible URL"/>
    <title type="text">My first document</title>
    <updated>2010-05-19T08:43:33Z</updated>
</entry>
```

This response can be used in the context of the creation of an external tool instance (section 3.5.1). It corresponds to the interaction 3.7 in Figure 3.2.

`GET /instance/{instanceId}` (response)

Responses to a `GET /instance/{instanceId}` call on a tool adapter have the same structure as responses to successful `POST /instance` requests. The only possible difference is the content of the `atom:updated` element, which should contain the date of the last modification of the resource. This date could be thus later than the creation date. This response can be employed in the context of the retrieval (use) of an external tool instance (section 3.5.2). It corresponds to the interaction 1.4 in Figure 3.3.

B.3 Methods of the GLUE! contract for VLE adapters

`GET /tools` (response)

The response to a `GET /tools` request issued by a VLE adapter includes a list of entries with information about the tools that are registered in the GLUElet Manager. This list is formatted in an `atom:feed` element, which contains an `atom:entry` element for each external tool. This `atom:entry` element includes information concerning the particular tool; information that is distributed in various parameters. Among these parameters, it is important to note the unique identifier representing the tool. This identifier is employed in subsequent requests, for instance to retrieve the configuration template, or to create instances of that tool. Table 5 shows the values that must be assigned to the direct descendants of the `atom:feed` element. The values of the `atom:entry` element are not specified in this table, but in Table 6, where significant GLUE! elements are marked in bold. An example of response sent by the GLUElet Manager is presented next:

Table 5: `atom:feed` assignments for `GET /tools` responses from the GLUElet Manager.

Element	*Value that must be assigned*
`atom:author/atom:name`	Fixed string "Group of Intelligent and Cooperative Systems (GSIC)".
`atom:id`	URI of the `/tools` resource in the GLUElet Manager.
`atom:link, with rel="self"`	Same value as in `atom:id`, as recommended in the Atom data format.
`atom:title`	Fixed string "List of tools (tool implementations)".
`atom:updated`	Date of the last modification of the information about external tools in the internal tool registry.

Table 6: `atom:entry` assignments for `GET /tools` responses from the GLUElet Manager.

Element	*Value that must be assigned*
`atom:author/atom:name`	Fixed string "Group of Intelligent and Cooperative Systems (GSIC)".
`atom:id`	URI of the `/tools/{toolId}` resource in the GLUElet Manager.
`atom:link, with rel="alternate"`[6]	Same value as in `atom:id`.
`atom:title`	**Tool name.**
`atom:updated`	Date of the last update of this tool in the internal tool registry.
`glue:impName`	**Kind of implementation under which the external tool is provided.**
`glue:toolProvider`	**Name of the provider of this tool.**
`atom:link, with rel="related"`	**Contact URL of the provider of this tool.**

[6] This element will be substituted in the near future by an `atom:content` element, which should include a textual description of the tool

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
    xmlns:glue="http://gsic.uva.es/glue/1.0">
    <author>
        <name>Group of Intelligent and Cooperative Systems (GSIC)</name>
    </author>
    <id>http://localhost:8185/GLUEletManager/tools</id>
    <link href="http://localhost:8185/GLUEletManager/tools" rel="alternate" title=""/>
    <title type="text">List of tools (tool implementations)</title>
    <updated>2010-05-12T06:55:10.37Z</updated>

    <entry>
        <author>
            <name>Group of Intelligent and Cooperative Systems (GSIC)</name>
        </author>
        <id>http://localhost:8185/GLUEletManager/tools/1</id>
        <link href="http://localhost:8185/GLUEletManager/tools/1" rel="alternate"
            title="Description of Google Documents"/>
        <link href="http://docs.google.com/" rel="related" title="Provider"/>
        <title type="text">Google Documents</title>
        <updated>2010-04-20T09:15:00Z</updated>
        <glue:impName>GData</glue:impName>
        <glue:toolProvider>http://docs.google.com/</glue:toolProvider>
    </entry>

(...)
</feed>
```

This response can be used in the context of the creation of an external tool instance (section 3.5.1). It corresponds to the interaction 1.3 in Figure 3.2.

`GET /configuration` (response)

The response to a `GET /configuration` call on the GLUElet Manager consists of a single `atom:entry` element, containing a configuration template following either the XForms or the HTML5 format. The values that must be assigned to each element in this response are the same that in the `GET /configuration` response given to the GLUElet Manager from a tool adapter. Only one element is changed by the GLUElet Manager. This element is `atom:id`, which must point at the URI of the */tools/{toolId}/configuration* resource in the GLUElet Manager domain. An example of the response sent by the GLUElet Manager is provided next:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<entry xmlns="http://www.w3.org/2005/Atom"
    xmlns:glue="http://gsic.uva.es/glue/1.0">
    <author>
        <name>Group of Intelligent and Cooperative Systems (GSIC)</name>
    </author>
```

```
    <id>http://localhost:8185/GLUEletManager/tools/1/configuration</id>
    <title type="text">Default configuration definition</title>
    <updated>2010-05-12T06:55:39.26Z</updated>
    <content type="xhtml">
    <div xmlns="http://www.w3.org/1999/xhtml"
        xmlns:ev="http://www.w3.org/2001/xml-events"
        xmlns:xf="http://www.w3.org/2002/xforms"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:ins="http://gsic.uva.es/glue/adaptors/implementation/gdata/1.0">
        <!-- model -->
        <xf:model>
            <xf:instance xmlns="" id="input-file">
                <ins:data>
                <ins:file xsi:type="xs:hexBinary"/>
                <ins:title xsi:type="xs:string">Default title</ins:title>
                </ins:data>
            </xf:instance>
            <xf:submission action="http://inapropiate.url.com/send" id="s"
                method="post"/>
        </xf:model>
        <!-- user interface control in body; only 'ref' bindings are used -->
        <p>
            <xf:input ref="/data/title">
               <xf:label>Title:</xf:label>
            </xf:input>
        </p>
        <p>
            <xf:upload ref="/data/file">
                <xf:label>Select file:</xf:label>
                <xf:filename ref="@filename"/>
                <xf:mediatype ref="@mediatype"/>
            </xf:upload>
        </p>
        <p>
            <xf:submit submission="s">
                <xf:label>Submit</xf:label>
            </xf:submit>
        </p>
    </div>
    </content>
</entry>
```

This response can be used in the context of the creation of an external tool instance (section 3.5.1). It corresponds to the interaction 2.5 in Figure 3.2.

`POST /instance` (request)

The HTTP entity that is sent in the requests to the GLUElet Manager for the creation of instances must contain an `atom:entry` element that includes, among other items, the configura-

tion values provided by the educator and the list of users. Table 7 shows the descendant elements of the `atom:entry`. Significantly, these elements are the same than in those requests from the GLUElet Manager to tool adapters except for the "*parameters*" element, which is generated by the GLUElet Manager. Besides, an example of this request can be seen next:

Table 7: `atom:entry` assignments for `POST /instance` requests to the GLUElet Manager.

Element	*Value that must be assigned*
`glue:tool`	**Parameter "tool", pointing at a URI in the GLUElet Manager. This URI indicates the tool for which the instance is created.**
`glue:configuration`	**Parameter "configuration", as an XML composed-element. Its content includes the configuration values provided by the educator, once validated.**
`glue:users`	**Parameter "users", as a list of `glue:user` elements. Each of these elements contains a textual identifier of a user (VLE username) that shares the external tool instance, once created.**
`glue:callerUser`	**Parameter "callerUser" containing a textual identifier (VLE username) of the user that requests the creation of the tool instance.**

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<entry xmlns="http://www.w3.org/2005/Atom"
    xmlns:glue="http://gsic.uva.es/glue/1.0">
    <glue:tool>http://localhost:8185/GLUEletManager/tools/1</glue:tool>
    <glue:configuration>
        <ins:data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:ins="http://gsic.uva.es/glue/adaptors/implementation/gdata/1.0">
            <ins:file filename="prueba.odt" xsi:type="xs:hexBinary">
                504(...)3600000000
            </ins:file>
            <ins:title xsi:type="xs:string">My first document</title>
        </ins:data>
    </glue:configuration>
    <glue:users>
        <glue:user>
            student1
        </glue:user>
        <glue:user>
            student2
        </glue:user>
        <glue:user>
            student3
        </glue:user>
        <glue:user>
            student4
        </glue:user>
    </glue:users>
    <glue:callerUser>
        teacher
    </glue:callerUser>
</entry>
```

This response can be employed in the context of the creation of an external tool instance (section 3.5.1). It corresponds to the interaction 3.2 in Figure 3.2.

`POST /instance` (response)

The response to a `POST /instance` returned by the GLUElet Manager consists on an `atom:entry` element with information of the tool instance that is created. Table 8 shows the descendants elements of this `atom:entry`. It is noteworthy that this response is originally generated by a tool adapter, and the GLUElet Manager only updates the `atom:id`. An example of this response after the creation of a Google Documents instances is shown next:

Table 8: `atom:entry` assignments for `POST /instance` responses from the GLUElet Manager.

Element	*Value that must be assigned*
`atom:id`	URI of the `/instance/{instanceId}` resource that has been created in the GLUElet Manager.
`atom:link, with rel="alternate"`	**URL of the real resource that has been created. It must be accessible from a web browser. The VLE adapter may use this parameter to show the instance that has been successfully created to the educator that requested its creation. This URL should not be stored by the VLE adapter, which should send a new request to the GLUElet Manager when a particular end-user need to access that instance.**
`atom:author`	Information about the tool adapter or about its developer. GLUE! built-in adapters include an `atom:author/atom:name` element with the string "Group of Intelligent and Cooperative Systems (GSIC)".
`atom:title`	Textual descriptor of the instance that is created. Its representation depends on the specific tool or tool adapter.
`atom:updated`	Date with the creation of the instance.

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom"
    xmlns:glue="http://gsic.uva.es/glue/1.0">
    <author>
        <name>Group of Intelligent and Cooperative Systems (GSIC)</name>
    </author>
    <id>http://localhost:8185/GLUEletManager/instance/1</id>
    <link
        href="http://docs.google.com/Doc?docid=0AUMiWAp0au_cZGdmend4cl8yNjFjNjcydGdmaA&amp;hl=en"
        rel="alternate" title="HTML accesible URL"/>
    <title type="text">Title for this instance</title>
    <updated>2010-05-19T08:43:33Z</updated>
</entry>
```

This response can be employed in the context of the creation of an external tool instance (section 3.5.1). It corresponds to the interaction 3.9 in Figure 3.2.

`GET /instance/{instanceId}` (response)

Responses to a `GET /instance/{instanceId}` call on the GLUElet Manager have the same structure as responses to successful `POST /instance` requests. There is only one difference in the content of the `atom:updated` element. In this case, this element must contain the date of the last modification of the instance, and so, this date could be later than the creation date. This response can be employed in the context of the retrieval (use) of an external tool instance (section 3.5.2). It corresponds to the interaction 1.5 in Figure 3.3.

Appendix C: Developing tool adapters in Java

In order to facilitate the development of new tool adapters in Java, a library is provided to be used by anyone interested in contributing to the GLUE! architecture. This library, called ***GLUEcommon***, helps developers to focus on the implementation of the code required for the communication with external tools, rather than on the HTTP messages received from the GLUElet Manager.

C.1 The *GLUEcommon* library

GLUEcommon is a library for Java programmers developed by the GSIC-EMIC research group. The structure of this library is as follows:

1. **Package** `glue.common.entities.configuration`. This package contains classes responsible for handling the configuration forms that must be provided by tool adapters. Besides, these classes also retrieve the values provided by educators after filling out these forms.
   - `BasicConfigurationRepository`: class providing access to all the configuration forms available in the tool adapter *classpath* as XHTML files.
   - `Configuration`: class representing configuration forms.
   - `ConfigurationRepository`: interface that allows developers to create their own mechanisms to access configuration forms. This interface is implemented by `BasicConfigurationRepository`.
2. **Package** `glue.common.format`. This package contains helper classes employed in formatting tasks. Nevertheless, those developing tool adapters do not need to know further details on these classes, since they are not directly used.

3. **Package** `glue.common.entities.instance`. This package contains several interfaces that must be implemented by those developing tool adapters, in order to provide the data concerning to the tool instances that are created. This package also includes mechanisms for the creation, access and persistence of instances.

   - `BasicInstanceEntityRepository`: class providing a basic persistence system for the data related to external tool instances.
   - `InstanceEntity`: interface to be implemented by instance representations.
   - `InstanceEntityFactory`: interface to be implemented in order to provide mechanisms for the creation of instances.
   - `InstanceEntityRepository`: interface that allows developers to create more sophisticated persistence mechanisms. This interface is implemented by `BasicInstanceEntityRepository`.

4. **Package** `glue.common.resources`. This is a package that includes several classes responsible for the reception and processing of requests from the GLUElet Manager.

   - `ConfigurationResource`: class processing requests related to configuration forms.
   - `GLUEResource`: ancestor class of all the other resource classes.
   - `InstanceFactoryResource`: abstract class processing requests for the creation of new tool instances.
   - `InstanceResource`: abstract class processing requests for the access, update and deletion of created instances.

5. **Package** `glue.common.server`. This package contains the classes in charge of the initialization of the tool adapter.

   - `Application`: abstract class to be implemented in order to map the resource classes responsible for processing the requests from the GLUElet Manager to the URL paths where each request is served.
   - `Server`: class in charge of the initial configuration of the server. It also starts the server process.

Figure 3 presents a class diagram that summarizes the main relationships between the different classes and interfaces in the *GLUEcommon* library.

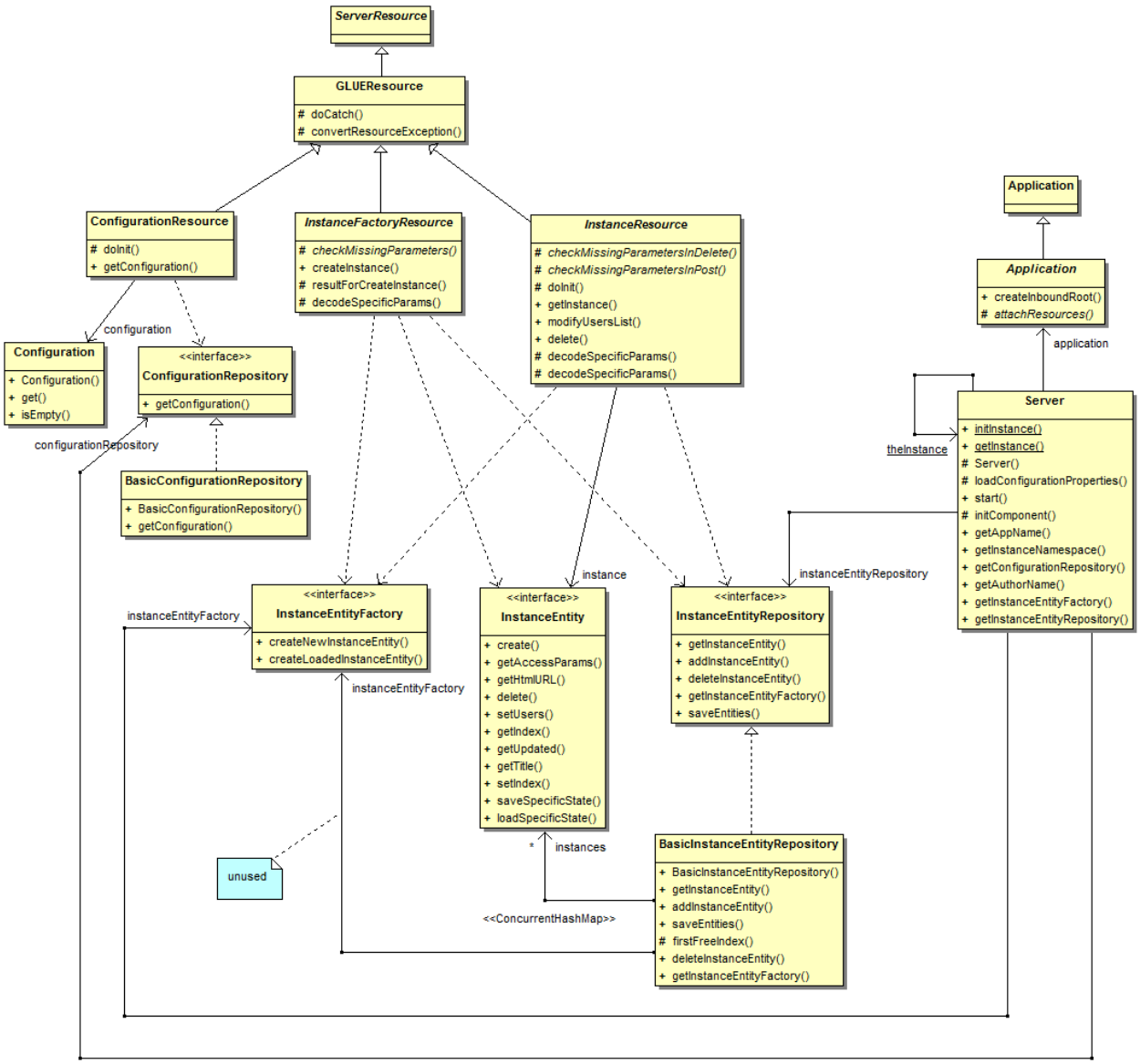

Figure 3: Class diagram representing the classes and interfaces in the *GLUEcommon* library.

C.2 Configurations

The GLUE! contract for tools assumes that the creation of tool instances may require the provision of specific values for certain configuration parameters that cannot be predicted by the contract itself. Some of these parameters are tied to the tool, and so they are common for every instance created (e.g. a parameter indicating if the instances of that tool can be reused in different learning activities). However, some other parameters may present different values for every instance created (e.g. a parameter to configure the title of the instance); these specific parameters should be provided by the educator during the creation of instances.

A configuration is thus defined as the set of generic and specific parameters, whose values need to be provided to succeed in the creation of a particular external tool instance. The definition of these parameters must be done by tool adapters, employing configuration templates. A configuration template is an XForms-in-XHTML form containing the names, types, and possibly default values for the generic and specific parameters that must be provided in order to create instances of a certain tool. Alternatively, HTML5 can be used for the definition of configuration forms, instead of XForms, as established in the GLUE! integration contracts.

The *GLUEcommon* library includes the classes `ConfigurationResource`, `Configuration` and `BasicConfigurationRepository` to implement the `GET /configuration` method defined in the GLUE! contract for tools. Developers implementing new adapters in Java only need to write the XForms-in-XHTML file for each intended configuration template (each tool may have a different configuration template, although several tools wrapped by the same tool adapter may share the same configuration template too). Files with configuration templates must be set in a directory included in the *classpath* of the tool adapter. All the tool adapters provided as examples have a *conf/* directory containing the configuration forms. Some minor changes must be made in the source code to access the configuration templates. These changes should be in the class with the `main` method, and are later detailed in this appendix.

XForms-in-XHTML files: example, details and limitations

Tool adapters must provide configuration templates using the XForms specification (or HTML5). VLE adapters must be able to process these configuration templates in those cases where web browsers cannot directly show their content. The next example should be taken as a model for the definition of XForms configuration templates in new tool adapters. This example corresponds to the tool adapter that enables the integration of Google Docs (Documents, Spreadsheets and Presentations) in VLEs. In this case, a title and an initial file are the specific parameters that can be configured when creating instances of these tools.

```
<div xmlns="http://www.w3.org/1999/xhtml"
    xmlns:xf="http://www.w3.org/2002/xforms"
    xmlns:ev="http://www.w3.org/2001/xml-events"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ins="http://gsic.uva.es/glue/adaptors/implementation/gdata/1.0"
    xmlns:glue="http://gsic.uva.es/glue">

    <!-- model -->
    <xf:model>
        <xf:instance id="input-file">
            <ins:data>
                <glue:reusable xsi:type="xs:boolean">true</glue:reusable>
                <ins:file xsi:type="xs:base64Binary" ins:filename="" ins:mediatype=""/>
```

```
                    <ins:title xsi:type="xs:string">Your title here</ins:title>
            </ins:data>
            </xf:instance>
            </xf:model>

        <!-- user interface control in body; only 'ref' bindings are used -->
        <p>
            <xf:input ref="/ins:data/ins:title">
                <xf:label>Title:</xf:label>
            </xf:input>
        </p>
        <p>
            <xf:upload ref="/ins:data/ins:file">
                <xf:label>Select file:</xf:label>
                <xf:filename ref="@ins:filename" />
                <xf:mediatype ref="@ins:mediatype" />
            </xf:upload>
        </p>
    </div>
```

There are three important parts in the example: the definition of XML *namespaces* in the root `<div>` element; the *data model* where the configuration parameters are specified; and the *user interface elements* that must be shown to the teacher in the VLE interface. Every configuration file defined using the *GLUEcommon* library must be contained in a root `<div>` XHTML element. The previous example presents in the `<div>` attributes all the *namespace* definitions that should normally included. It is important to copy these attributes (including the prefixes associated to every *namespace*), in order to ensure the compatibility with VLE adapters. The only *namespace* that must be replaced is the value of `xmlns:ins`, which indicates the location of the parameters. Any URL representing the developer's organization or the tool (or tools) supported by the configuration form may be used.

In order to define the configuration parameters, `<xf:model>`, `<xf:instance>` and `<ins:data>` should be kept as in the example. The XForms specification allows to define more than one model, instance or data. Nevertheless, due to compatibility reasons with other existing adapters, only one of each kind is advised. On the contrary, the content of `<ins:data>` supports as many XML elements as needed. Each XML element represents a parameter that can be configured in each tool instance. Remarkably, the `ins` prefix must be used in these elements. Also, the `xsi:type` defines the type of attribute. Apart from those attributes illustrated here, other non-standard attributes may be freely added and bound to user interface elements.

There is a special predefined element that can be included in the content of `<ins:data>`. This element is the `<glue:reusable>` boolean flag, which indicates that the instances of this tool can be reused in different activities (i.e. the external tool supports the update of users sharing instances). This is one generic configuration element for all the instances created with

this adapter, and so, it must not be bound to the user interface (this value is not determined by the educator, but by the developer of the tool adapter).

The user interface elements are defined using XHTML. Though browsers may render all the elements defined in the XHTML specification, current VLE adapters only process two kinds of XHTML elements: `<xforms:input>` (text field) and `<xforms:upload>` (file attachment). Future improvements of VLE adapters are intended to process additional XHTML elements. The data type in the `<xforms:input>` elements is not validated in VLEs as of this writing. Therefore, the parsing of the values under this element corresponds to the tool adapter (as part of the process of creating instances). Regarding the `<xforms:upload>` elements, only text codifications are currently supported. In order to associate the user interface elements and the data model, the `ref` attribute should be included in the input elements, specifying a path that follows the XML Path Language (XPath) inside the `<xf:instance>`.

C.3 Instances

The interfaces `InstanceEntityFactory` and `InstanceEntity` must be implemented in different classes to manage external tool instances. `InstanceEntityFactory` aims at creating `InstanceEntity` objects, which are the local representation of actual instances in the memory space of the tool adapter.

The `InstanceEntity` interface is detailed next:

```
public interface InstanceEntity {
    public void create(String callerUser, Map<String,String> specificParams)
        throws ResourceException;
    public String getAccessParams(String callerUser, Map<String,String> specificParams);
    public String getHtmlURL(String callerUser, Map<String,String> specificParams);
    public String delete(Map<String, String> specificParms) throws ResourceException;
    public void setUsers(List<String> users, String callerUser,Map<String, String> specificParams);
    public int getIndex();
    public Date getUpdated();
    public String getTitle();
    public void setIndex(int index);
    public void saveSpecificState(PrintStream out);
    public void loadSpecificState(BufferedReader in) throws IOException;
}
```

The `InstanceEntityFactory` defines the following methods to be implemented:

```
public interface InstanceEntityFactory {
    public InstanceEntity createNewInstanceEntity(String toolName, List<String> users,
        String callerUser, Map<String,String> specificParams, Element configuration);
    public InstanceEntity createLoadedInstanceEntity(int index, String title, Date updated);
}
```

Instance creation

The creation of new instances should be done in two different steps and involves several methods. These two steps are detailed next including a description of the following methods:

1. `InstanceEntityFactory.createNewInstanceEntity(...)`. The first step is the creation of a new class that implements the `InstanceEntityFactory` interface (let it be `MyInstanceEntityFactory`). The first method that must be provided, `createNewInstanceEntity(...)`, is called when the tool adapter receives a `POST /instance` request from the GLUElet Manager. The parameters of this method are described next:

```
public InstanceEntity createNewInstanceEntity(String toolName, List<String> users,
    String callerUser, Map<String,String> specificParams, Element configuration);
```

   - ***toolName***. This is the name of the tool for which a new instance is requested. The string value that identifies this tool is established by the developer of the tool adapter. This value must be documented, and should also be reported to the GLUElet Managers employed as clients of this tool adapter.
   - ***users***. This is the list of users (VLE usernames) that can access the new instance. There are two ways of dealing with this parameter: if the external tool has no knowledge about users or access control, then this parameter should be avoided; if the external tool can support different users or access control, then this list of users should be used in the communication between the tool adapter and the external tool to indicate that all these users should share the same instance.
   - ***callerUser***. This parameter refers to the user that requests the creation of a new instance. The *callerUser* is also included in the *users* parameter (list of users sharing the same external tool instance) in creation requests.
   - ***specificParams***. This parameter represents a list of key-value pairs where the information required for the creation of tool instances is received as part of a GLUElet Manager request. The kind of information that is expected here must be documented by the developer implementing the tool adapter, including the keys and values. These values are always the same for every GLUElet Manager that requests the creation of external tool instances. As an example, a common parameter that may be documented here is the URL of the service where the external tool provides its programmatic interface. Remarkably, those parameters that are specific for each instance should not be defined here, but in the configuration templates.

- ***configuration***. This is the set of values established by the educator after filling out the configuration form. These values are `org.w3c.dom.Element` objects included as children of the `<ins:data>` element from the XML formatted answer. Other libraries apart from the `org.w3c.dom` API may be used here in order to retrieve the values provided by the educator from the XML tree.

The final target of this method is the creation of an object of a new class that implements the `InstanceEntity` interface (let this class be `MyInstanceEntity`). The attributes and the structure of this class can be freely specified. A different `MyInstanceEntity` class might be defined for each external tool supported by the tool adapter, or a common `MyInstanceEntity` class might wrap all these tools. Different constructor methods can be defined in each class to be requested depending on the values sent by the educator in the configuration form.

2. `InstanceEntityFactory.create(...)`. The second step for the creation of a new instance is the call to the `create(...)` method of `MyInstanceEntity`. The implementation provided for this method must support the actual creation of new instances, and so, the communication between the tool adapter and the external tool for this purpose. This method should save the URLs representing the instances that are created. These URLs will later be retrieved by the GLUElet Manager when some of the VLE users want to access these instances. These URLs are returned by invoking the `MyInstanceEntity.getHtmlURL(...)` method, showing that the creation process has been accomplished. It is noteworthy that if the external tool does not return URLs representing tool instances, then the tool adapter must wrap these instances to provide that URLs itself (e.g. hosting these instances in its domain).

```
public void create(String callerUser, Map<String, String> specificParams)
    throws ResourceException;
```

3. `InstanceEntityFactory.getHtmlURL(...)`. When the creation of an instance is accomplished, the `InstanceEntity.getHtmlURL(...)` method is called in order to return the URL representing the corresponding external tools instance. The class `MyInstanceEntity` must thus provide an implementation of this method, whose response includes such URL. This URL can be saved as part of the `MyInstanceEntity` state, but it can also be stored in a local file of database. Rarely, it might be retrieved from the tool with each call to this method, although that would be less efficient.

```
public String getHtmlURL(String callerUser, Map<String,String> specificParams);
```

4. `InstanceEntityFactory.getAccessParams(...)`. This method returns to the GLUElet Manager the information that must be saved for future accesses to each external tool instance. This information could be a subset of the *specificParams* parameter, or could be new values generated for every new instance that are returned as keys to authorize the access in future requests.

```
public String getAccessParams(String callerUser, Map<String, String> specificParams);
```

Retrieval, update and deletion of instances

Once the creation of external tool instances is completed, tool adapters can receive other requests from the GLUElet Manager (as established in the GLUE! contract for tools) in order to retrieve, update or delete these instances. These requests are first processed by an `InstanceResource` object that deals with the details of the HTTP communications, translating these requests into calls to the methods defined in the `InstanceEntity` interface:

1. `InstanceEntityFactory.getHtmlURL(...)`. Requests to this method are triggered as the result of a `GET /instance/{instanceId}`, request. The implementation of this method has already been detailed in this section.

2. `InstanceEntityFactory.delete(...)`. When the tool adapter receives a `DELETE /instance/{instanceId}` request, then a request to the `InstanceEntity.delete(...)` method is sent. The class `MyInstanceEntity` must provide an implementation of this method that performs an interaction with the tool to destroy the instance. The deletion of the persisted data related to this instance is automatically done after the successful termination of this method. This method receives one parameter *specificParams*, which is a list of key-value pairs, in which the values returned at creation time by the `getAccessParams(...)` method regarding such instance are included.

```
public String delete(Map<String, String> specificParams) throws ResourceException;
```

3. `InstanceEntityFactory.setUsers(...)`. The reception of a `POST /instance/{instanceId}` triggers a request to the `InstanceEntity.setUsers(...)` method in the tool adapter. The class `MyInstanceEntity` must provide an implementation of this method that performs all the actions required to update the list of users sharing the instances represented by objects of this class. This could include interactions with the external tool, updating the state of the instance in the tool adapter, or both, depending upon the contract of the corresponding tool. The local state of the instance is automatically updated

after a successful completion of the `setUsers(...)` method. This method receives some parameters that may need a further explanation:

```
public void setUsers(List<String> users, String callerUser,
    Map<String, String> specificParams);
```

- ***users***. This is the new list of VLE usernames authorized to access the tool instance. This list also includes the *callerUser* as an additional element. Remarkably, those developing tool adapters are not required to provide a full authorization mechanism based on the list of users, but they must, at least, guarantee that the users in this list will not get a denial of access when trying to reach the tool instance later.
- ***callerUser***. This is the VLE-context name of the user trying to access the instance.
- ***specificParams***. This is a list of key-value pairs that includes the values returned at creation time by the `getAccessParams(...)` method of the same instance.

Instances data persistence

This section deals with the information about the instances that tool adapters need to permanently save in their local scope. Of course, those developing tool adapters can save all the data they need to manipulate tool instances, although some of these data must always be kept.

1. *Mandatory data fields.* The format applied to the data included as parameters or results of the GLUE! contract operations establishes some mandatory fields that must be included in the attributes of every tool instance. These fields are: an identifier, a title and an update date. The identifier is a URL, which is generated by the *GLUECommon* library. This URL is built using the domain and the port where the tool adapter listens to requests, the structure of resources defined by the GLUE! contract, and an index generated by the persistence system to identify the new instance that is created. This index must be saved as an attribute of the class implementing the `InstanceEntity` interface. The methods `getIndex()` and `setIndex(...)` must also be implemented in this class to retrieve and assign this index. The title is a text string that must be returned as a result of the `getTitle()` method. An attribute containing a title makes sense for many tool instances, and so it should be returned by this method. In those instances where a title is not defined, any text string could fit. The update date should be created by every constructor method of the `InstanceEntity` implementation class, being kept as an object attribute. The value of this attribute should be updated after a successful request to the `setUsers(...)` method. The accuracy of the value for the update date is not critical at all. Nevertheless, an implementation must be provided to the `getUpdated()` method of `InstanceEntity` that returns a value representing the update date.

2. *Saving other data.* The mandatory data described in the previous paragraphs are automatically handled by the `InstanceEntityRepository` object when persistence is required. Nevertheless, tool adapters might need to store additional data to carry out different operations with the external tools. The methods `saveSpecificState(...)` and `loadSpecificState()` are provided in the `InstanceEntity` interface to store and retrieve this special data. Both methods receive a single parameter (a `java.lang.PrintStream` to save or a `java.io.BufferedReader` to load) indicating the data that must be stored or retrieved. Developers should carefully implement these two methods to guarantee that the same values, formats, and order are consistently applied.

3. *Other methods.* The `InstanceEntiyFactory` interface includes a method, `createLoadedInstanceEntity(...)`, which has not been addressed yet. This method connects the entity repository handler with the constructor required to load the created and stored tool instances (if any) when the tool adapter is started. This method includes as parameters the aforementioned mandatory data fields (index, title, updated). The only operation expected from those implementing this method is a call to a constructor of the `MyInstanceEntity` class, passing the three mandatory data fields in the same way that they are received, as illustrated in this piece of code:

```
public InstanceEntity createLoadedInstanceEntity(int index, String title, Date updated) {
    return MyInstanceEntity(index, title, updated);
}
```

C.4 Resources and server

Most of the code required to listen and process requests from the GLUElet Manager is covered by the *GLUEcommon* package. Even so, some new code must be added to complete the server life cycle. Developers are strongly recommended to review the code of the existing tool adapters to get detailed examples for the next subsections.

1. *Checking specific parameters.* Several methods at `InstanceEntity` and `InstanceEntityResource` need a *specificParams* parameter containing key-value pairs of *string* type. The values for these pairs are not defined in the GLUE! contract, and so, each new tool adapter can define their owns. The documentation of a new tool adapter should include a list of keys and expected values, so that the GLUE! administrator can properly configure the internal tool registry. The checking of these parameters can be performed by the classes implementing the `InstanceEntity` and `InstanceEntityFactory` interfaces. Nevertheless, an early checking is preferred and better fits in the *GLUEcommon*

operation sequence. Several abstract methods are provided for this purpose, like the `checkMissingParameters(...)` in the `InstanceFactoryResource` class.

```
protected abstract String checkMissingParameters(String toolName,
    List<String> users, String callerUser,
    Map<String,String> specificParams);
```

Therefore, developers must create a class extending `InstanceFactoryResource` to provide the implementation for this method. The parameters received must contain the same values that those in the `InstanceEntityFactory.createNewInstanceEntity(...)` method, previously explained here. The expected result is a text string with a comma separated list with the names of wrong or missed parameters inside *specificParams* (or *null* in case of a successful checking). The `InstanceEntityFactory` object generates an HTTP 400 response in the event that the returned value is not *null*.

Two other abstract methods with an equivalent return value are provided in the `InstanceResource` class, `checkMissingParametersInDelete(...)` and `checkMissingParametersInPost(...)`.

```
protected abstract String checkMissingParametersInDelete(
    Map<String, String> specificParams);
protected abstract String checkMissingParametersInPost(String callerUser,
    List<String> users, Map<String, String> specificParams);
```

Other resource classes extending `InstanceResource` must be created to provide implementations of both checking methods. The first one is called before `InstanceEntity.delete(...)`, while processing a `DELETE /instance/{instanceId}` request; the second one is executed just before `InstanceEntity.setUsers(...)`, while processing a `POST /instance/{instanceId}` message. It is noteworthy that, in both cases, the values in *specificParams* must match with those returned by the method implementing `InstanceEntity.getAccessParams(...)` in the tool adapter.

2. *Map resources to URLs.* The classes extending `InstanceFactoryResource` and `InstanceResource` described in the previous paragraphs must be associated with the URLs of the resources they implement, as described in the GLUE! contract. To do so, a new class extending the `Application` abstract class in the *GLUEcommon* package must be defined. This new class must provide an implementation for the `attachResources(...)` method. The implementation is really simple and only requires the replacement of the `MyInstanceFactoryResource`, `MyInstanceResource`, and `MyApplication` class names in the next piece of code. It is very important to avoid any modification in the first parameter passed in the `attach(...)` calls, in order to meet the GLUE! contract for tools.

```
public class MyApplication extends Application {
    @Override
    protected void attachResources(Router router) {
    // Defines a route for the resource instance factory
    router.attach("/instance", MyInstanceFactoryResource.class);
    // Defines a route for the resource instance
    router.attach("/instance/{instanceId}", MyInstanceResource.class);
    // Defines a route for the resource configuration (default one)
    router.attach("/configuration", ConfigurationResource.class);
    // Defines a route for the resource configuration (required one)
    router.attach("/configuration/{configId}", ConfigurationResource.class);
    }
}
```

3. *Main class.* The last class that must be created is the first to be executed. This class must include a static `main(...)` method to initialize the application scope objects and to start listening requests from the GLUElet Manager. The next piece of code illustrates the steps to be included in the `main(...)` sequence.

```
public class MyApplication extends Application {
    ConfigurationRepository configurationRepository  =
        new BasicConfigurationRepository(
            toolImplementationNames, configurationIds,
            configurationFileNames);            // step 1

    InstanceEntityFactory instanceEntityFactory =
        new MyInstanceEntityFactory();          // step 2

    instanceEntityRepository =
        new BasicInstanceEntityRepository(
        instanceEntityFactory, "instances.txt");// step 3

    server = Server.initInstance(
        APP_NAME, AUTHOR_NAME, new MyApplication(),
        BASE_URL, CFG_FILENAME, INSTANCE_NAMESPACE,
        configurationRepository, instanceEntityFactory,
        instanceEntityRepository);              // step 4

    server.start();                             // step 5
}
```

The first step in the sample code is the initialization of the basic implementation for `ConfigurationRepository`, `BasicConfigurationRepository`. This object keeps in memory a list of triples, each of them containing the name of an external tool wrapped by the tool adapter, a numeric identifier (formatted as a decimal text string), and the name of an XForms-in-XHTML file contained in the *classpath* defining the configuration form for this external tool. This list is used to access the appropriate configuration form when

a `GET /configuration` request is received by the tool adapter. The parameters passed to the `BasicConfigurationRepository` are three vectors with the same number of text string elements, containing tool names, identifiers and configuration files.

The second step is the creation of an object of the `MyInstanceEntityFactory` class. This object is later employed for the creation of tool instances as the result of receiving `POST /instance` requests. The step three shows the creation of a global-scope entity repository handler. This handler must be an object of the class implementing the `InstanceEntityRepository` interface. This object is responsible for persisting the data of the instances, locally stored by the tool adapter. In this example, an object of the `BasicInstanceEntityRepository` class (included in *GLUEcommon*) is created for this purpose. Two arguments are passed to its constructor: a reference to the `InstanceEntityFactory` object created in the second step, and the name of a text file used to keep the data stored in the file system. Interestingly, the `BasicInstanceEntityRepository` is provided as a very simple solution, although more advanced entity repository classes might be developed for other tool adapters.

A `Server` object is created in the fourth step, being started in the last step (`server.start()`). This object is initialized with references to the objects instantiated in the steps one to three, a new `MyApplication` object, and some text values that can be customized for each tool adapter:

- **APP_NAME**. A name for the tool adapter; for example, "*GSIC Tool Adapter*".
- **AUTHOR_NAME**. A name identifying the author of the adapter; for example, "*Group of Intelligent and Cooperative Systems (GSIC)*".
- **BASE_URL**. A URL segment, common to all the resources in the tool adapter. This URL must start with `/ToolAdapter` plus any valid URL segment to complete the string. The BASE_URL segment is inserted in the structure `http://[host][BASE_URL]/[RESOURCE_SEGMENT]`. An example of a URL to an instance resource in a tool adapter with BASE_URL = "*GSICAdapter*" could be: `http://localhost:8888/ToolAdapter/GSICAdapter/instance/15`.
- **CFG_FILENAME**. The name (without the *.properties* extension) of a Java properties file where the tool adapter searches for property values at start time. The properties file must be in the tool adapter *classpath* to be found by the `Server` object. A property *port* with an integer value for the port where the adapter listens to requests must be included in the properties file. As an example, the tool adapters developed by the GSIC group use the *app* name to locate the *app.properties* files contained in the *conf/* directory.

- **INSTANCE_NAMESPACE**. An XML *namespace* that must coincide with the *namespace* used for the `<data>` element in the configuration forms returned by the tool adapter as the result of `GET /configuration` requests. For example, `http://gsic.uva.es/glue/adapters/implementation/dabbleboard/1.0`. The *namespace* does not need to match with any specific structure, but *namespaces* starting with `http://gsic.uva.es` should be avoided.

Appendix D: Installation and configuration manuals

GLUE!-RI, the reference implementation of the GLUE! architecture has been presented in chapter 4. GLUE!-RI includes a reference implementation of the GLUE! core, nine tool adapters and three VLE adapters. This appendix presents the installation and configuration manuals for all this elements.

D.1 Installation and configuration of the GLUE! core

Preliminary notes

The binary distribution of the GLUElet Manager reference implementation has been packaged and tested for its use in GNU/Linux and Windows (XP, Vista or 7) systems. However, as a Java Standard Edition 6 (JSE6) program, it could probably run in other operating systems where a JSE6 runtime platform is available. Even so, this document describes the installation and configuration processes just for Linux and Windows systems, highlighting their differences when needed. Administration permissions are required to install and configure the GLUElet Manager in any operating system. In Windows Vista and Windows 7, the command console must be started in administrator mode (right click, then "*Execute as administrator*").

It is noteworthy that the contents of the GLUElet Manager binary distribution are designed to coexist with other GLUE! components developed as JSE6 programs. In this section, the term `$GLUE_HOME` is used to refer to the base directory where all the JSE6 GLUE! components are to be located. Despite this, there are no real dependencies between the different components at the file system level. Therefore, files could be moved to other locations if needed, checking the runnable scripts in `$GLUE_HOME/bin` and `$GLUE_HOME/[component]/bin` directories to find out how each component should be started[7].

[7] It should be noted that the character "/" is used as a directory separator independently of the operating system.

Dependencies

The GLUElet Manager is a JSE6 program, and so, a JSE6 runtime context must be installed in the system where the GLUElet Manager is intended to run. Besides, the GLUElet Manager needs a SQL database system to keep information about the external tools that are integrated, and to provide persistence for the created instances. Any common database system with a JDBC driver available should be suitable. The JDBC driver must be obtained from the database system provider to grant the GLUElet Manager access to its databases. Additionally, the GLUElet Manager has the following dependencies:

- *GLUEcommon 0.8*. This is the library with the common code for GLUE! Java components. This library is located at `$GLUE_HOME/lib`.
- *RESTlet 2.0RC4 for Java Standard Edition* (only *org.restlet.jar*, *org.restlet.ext.atom.jar* and *org.restlet.ext.xml.jar* files). This is the Restlet library. By now, we discourage administrators from using any other different Restlet version here. This library is located at `$GLUE_HOME/lib/dep/restlet-jse-2.0.4`.
- *API and implementation of Java Persistence API 2 (JPA2)*. The *EclipseLink 2.0.0* is provided for persistence, although any other JPA2 implementation could be used instead. This library is located at `$GLUE_HOME/lib/dep/eclipselink`.

All these dependencies are included in the GLUElet Manager binary distribution as *jar* files in different subdirectories. The `README.txt` file contains more information about each library and its original provider.

Installation

The steps that must be followed in order to install the GLUElet Manager are:

1. Choose and create the `$GLUE_HOME` directory. For instance, in Linux, the directory `/usr/share/glue` could be a suitable path for `$GLUE_HOME`.
2. Define the system variable `$GLUE_HOME` containing the full path pointing at the chosen `$GLUE_HOME` directory: in Windows, click on "*Start*" button, "*Control Panel*", "*System and Maintenance*", "*System*", "*Advanced system*" settings, "*Advanced*" tab, "*Environment Variables*" button; in Linux, add `export $GLUE_HOME=/usr/share/glue` in `/etc/bash.bashrc` or `/etc/profile`.
3. Extract the files from the binary package at `$GLUE_HOME`. Then, the files presented in Table 9 are added in `$GLUE_HOME`.

Table 9: Content extracted from the binary package in the GLUElet Manager.

Directory under $GLUE_HOME	File	Purpose
bin/	ServiceInstaller.exe	Apache Foundation helper to register Windows services
lib/	glue-common.jar	Common elements for GLUE! Java components
lib/dep/eclispselink/	eclipselink.jar	JPA2 implementation by Eclipse Foundation
lib/dep/eclispselink/	javax.persistence_2.0.0.v200911271158.jar	JPA2 implementation by Eclipse Foundation
lib/dep/restlet-jse-2.0rc4/	org.restlet.ext.atom.jar	RESTlet selected files
lib/dep/restlet-jse-2.0rc4/	org.restlet.ext.xml.jar	RESTlet selected files
lib/dep/restlet-jse-2.0rc4/	org.restlet.jar	RESTlet selected files
manager/bin/	install-gm.sh	Installs the GLUElet Manager in Linux
manager/bin/	install-gm.bat	Registers the GLUElet Manager as a Windows service
manager/bin/	start-gm.sh	Starts the GLUElet Manager in Linux
manager/bin/	stop-gm.sh	Stops the GLUElet Manager in Linux
manager/bin/	uninstall-gm.sh	Cleans the GLUElet Manager installation in Linux
manager/bin/	uninstall-gm.bat	Removes the GLUElet Manager from the Windows services register
manager/conf/	app.properties	Runtime configuration file
manager/conf/db	create_GLUE_databases.sql	Database helper script
manager/conf/db	drop_GLUE_databases.sql	Database helper script
manager/conf/db	SQL-2003-notes.txt	Database helper script
manager/conf/META-INF/	persistence.xml	Database access file
manager/lib/	gluelet-manager.jar	GLUElet Manager binary files
manager/log/	*.log	Log files
manager/	COPYING.txt	GNU GPL license
manager/	INSTALL.txt	Installation manual
manager/	README.txt	GLUElet Manager license and third-party notices

4. In Linux, set execution permissions to the `*.sh` files in `$GLUE_HOME/manager/bin`. To do so, the following command must be executed: `chmod +x $GLUE_HOME/manager/bin/*.sh`.

5. Place a copy of the JDBC drivers of the system in `$GLUE_HOME/lib/dep/jdbc-connector/` (as one or multiple *jar* files).

6. In Linux, run `$GLUE_HOME/manager/bin/install-gm.sh`. The execution of that file: creates the empty log file `/var/log/glue/manager.log`, and a symbolic link pointing at this file in `$GLUE_HOME/manager/log`; and creates symbolic links at `/usr/bin` pointing at `$GLUE_HOME/manager/bin/start-gm.sh` and `$GLUE_HOME/manager/bin/stop-gm.sh`.

In Windows, run `$GLUE_HOME/manager/bin/install-gm.bat`. The execution of that file: registers the GLUElet Manager as a Windows service; and defines `$GLUE_HOME/manager/log/manager.log` as the log file (it is created with the first start of the GLUElet Manager). By default, the GLUElet Manager is registered to be started at system boot time. The parameter *manual* should be passed to `install-gm.bat` in order to force the manual start of the GLUElet Manager.

Creation, configuration and access to databases

The GLUElet Manager requires the existence of two separate databases. The directory `$GLUE_HOME/manager/conf/db` contains a SQL script file, `create_GLUE_databases.sql`, which defines the structure of both databases. The first database, the *internal tool registry*, keeps the information needed to integrate every available external tool in GLUElet Manager clients (typically VLEs or other platforms). The system administrator must populate the internal tool registry with proper values for each tool. The "*Internal Registry*" section in the `INSTALL.txt` file of each existing tool adapter gives more information about these values. Otherwise, further details about these values should be requested to those developing or providing (if externally hosted) tool adapters. The second database is the *gluelets repository*. The GLUElet Manager stores in the *gluelets* repository the data about the instances of the registered tools created from its clients.

The `$GLUE_HOME/manager/conf/db/create_GLUE_databases.sql` must be executed in the management console of the database system in order to create both databases. The file `$GLUE_HOME/manager/conf/db/drop_GLUE_databases.sql` executes the operations to erase both databases. It is important to note that this operation destroys all the objects that are created by the GLUElet Manager.

Finally, the file `$GLUE_HOME/manager/conf/META-INF/persistence.xml` must be edited, in order to provide the details of the JDBC connection with the database system. More specifically, the "`value`" attribute of the following properties must be edited as required (it should be noted that all these properties appear twice in the `persistence.xml` file):

- `<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/InternalRegistry"/>`. This is the URL to access each database. It includes the name of the database at the end (`InternalRegistry` or `GLUEletsRepository`).
- `<property name="javax.persistence.jdbc.user" value="glue"/>`. This is an authorized database user.
- `<property name="javax.persistence.jdbc.password" value="glue"/>`. This is the password of the authorized database user.

- `<property name=“javax.persistence.jdbc.driver” value=“com.mysql.jdbc.Driver”/>`. This is the name of the JDBC driver class in the documentation of that system.

Significantly, the password is saved in plain text. Thus, it is strongly recommended to create a specific user for the sole purpose of the GLUElet Manager operation. This user needs, at least, read permissions over all the internal tool registry tables, and read and insertion permissions over the *gluelets* repository tables.

Operation

Before starting the GLUElet Manager, the file `$GLUE_HOME/manager/conf/app.properties` can be edited to change some runtime properties:

- *port*. This is the number of the port where the GLUElet Manager listens for requests from VLE adapters.
- *logging*. This parameter must be “*on*” to save information in the log file about each request processed by the GLUElet Manager.

Besides, to manually start and stop the GLUElet Manager, it is recommended to follow these instructions:

- In Linux, use the commands `start-gm` and `stop-gm`.
- In Windows, access to the service management panel (“*Start*” button, “*Control Panel*”, “*System and Maintenance*”, “*Administration Tools*”, “*Services*”), search and select the service with the name “*GLUEletManager*” in the service list, and then click the “*Init*” or “*Stop*” links at the top left corner of the list.

After that, the content of the `$GLUE_HOME/manager/log/manager.log` can be checked to see if the GLUElet Manager started without problems. In that case, a couple of messages should appear similar to the following:

```
24-oct-2011 10:16:09 org.restlet.engine.http.connector.HttpClientHelper
start INFO: Starting the default HTTP client

24-oct-2011 10:16:14 org.restlet.engine.http.connector.HttpServerHelper
start INFO: Starting the default [HTTP/1.1] server on port 8185
```

Final notes

Some final notes must be taken into account when installing and configuring the GLUElet Manager:

1. If the access to the GLUElet Manager must be restricted to a limited set of system users for its start/stop, then it is important to keep in mind that authorized users need: execution permission on `$GLUE_HOME/manager/bin/*.sh`; read permissions on `$GLUE_HOME/manager/conf/app.properties`, `$GLUE_HOME/manager/conf/META-INF/persistence.xml` and all the *jar* files; and write permissions on `/var/log/glue/manager.log`.
2. The GLUElet Manager databases may be hosted by different providers, rather than only by the one in which the GLUElet Manager is installed. Nevertheless, the two databases need to be in the same database management system.
3. In Windows systems, every time the GLUElet Manager is started, the log file is erased.

D.2 Installation and configuration of tool adapters

This section presents the generic installation and configuration process for tool adapters. The specific particularities of tool adapters should be checked in the `INSTALL.txt` file of each adapter, which is located at `$GLUE_HOME/tool_adapters/{adapter}/log`. Significantly, the parameter `{adapter}` is used along this section to embrace any of the available tool adapters. This parameter must be replaced by the correspondent name of the tool adapter in each case, e.g. `dabbleboard`, `doodle` or `mediawiki`.

Preliminary notes

The binary distribution of the nine tool adapters in the reference implementation (Google Docs, MediaWiki, Dabbleboard, Apache Wookie, Doodle, Facebook Live Stream, Kaltura, Noteflight and web content) has been packaged and tested for their use in GNU/Linux and Windows (XP, Vista or 7) systems. However, as Java Standard Edition 6 (JSE6) programs, they will probably run in other operating systems where a JSE6 runtime platform is available. This document only describes the generic installation and configuration processes of tool adapters for Linux and Windows systems, highlighting their differences when needed. Administration permissions are required to install and configure tool adapters in any system. In Windows Vista and Windows 7, the command console must be started in administrator mode (right click, then "*Execute as administrator*").

It is noteworthy that the contents of tool adapter binary distributions are designed to coexist with other GLUE! components developed as JSE6 programs. As in the previous section, the term `$GLUE_HOME` is employed to refer to the base directory where all the JSE6 GLUE! components are to be located, although files could be moved to other locations if needed (see the previous section for further details).

Dependencies

The nine tool adapters in GLUE!-RI are JSE6 programs, and so, a JSE6 runtime context must be installed in the system where each of them is going to be installed. Besides, the nine tool adapters have the following dependencies:

- *GLUEcommon 0.8.* This is the library with the common code for GLUE! Java components. This library is located at `$GLUE_HOME/lib`.
- *RESTlet 2.0RC4 for Java Standard Edition* (only *org.restlet.jar*, *org.restlet.ext.atom.jar* and *org.restlet.ext.xml.jar* files). This is the Restlet library. The use of different Restlet versions is not recommended as of this writing. This library is located at `$GLUE_HOME/lib/dep/restlet-jse-2.0.4`.

In addition, there exist other dependencies in some of the adapters:

- *Wookie Connector Framework* (**Apache Wookie adapter**). This library is provided to get access to Apache Wookie servers. It is located at `$GLUE_HOME/tool_adapters/wookiewidgets/lib/dep/wookie-connector-framework`.
- *Google Data Client Library for Java 1.41.1* (**Google Docs adapter**). This library is provided to get access to Google Documents, Spreadsheets and Presentations. It is located at `$GLUE_HOME/tool_adapters/gdata/lib/dep/gdata-1.41.1`.
- *Java Mail API 1.4.3* (**Google Docs adapter**). The *Google Data Client Library for Java 1.41.1* uses this library, which is located at `$GLUE_HOME/tool_adapters/gdata/lib/dep/javamail-1.4.3`.

Generic dependencies are included in each tool adapter binary distribution as *jar* files in different subdirectories. Specific dependencies are only included in the subdirectories of the corresponding tool adapters (also as *jar* files). The `README.txt` file of each tool adapter gives further information about each library and its original provider. The `README.txt` file also details the information that must be set in the internal tool registry, regarding the external tools wrapped by each adapter.

Installation

The steps that must be followed in order to install the any tool adapter are:

1. Choose and create the `$GLUE_HOME` directory (e.g. `/usr/share/glue` in Linux).
2. Define the system variable `$GLUE_HOME` containing the full path pointing at the chosen `$GLUE_HOME` directory. The steps to define a system variable in both Windows and Linux have already been described in this appendix.
3. Extract the files from the binary package of the tool adapter at `$GLUE_HOME`. Then, the files presented in Table 10 are added in `$GLUE_HOME`.

Table 10: Content extracted from the binary package in tool adapters.

Directory under $GLUE_HOME	File	Purpose
bin/	ServiceInstaller.exe	Apache Foundation helper to register Windows services
lib/	glue-common.jar	Common elements for GLUE! Java components
lib/dep/restlet-jse-2.0rc4/	org.restlet.ext.atom.jar	RESTlet selected files
lib/dep/restlet-jse-2.0rc4/	org.restlet.ext.xml.jar	RESTlet selected files
lib/dep/restlet-jse-2.0rc4/	org.restlet.jar	RESTlet selected files
tool_adapter/{adapter}/bin/	install-{adapter}.sh	Installs the tool adapter in Linux
tool_adapter/{adapter}/bin/	install-{adapter}.bat	Registers the tool adapter as a Windows service
tool_adapter/{adapter}/bin/	start-{adapter}.sh	Starts the tool adapter in Linux
tool_adapter/{adapter}/bin/	stop-{adapter}.sh	Stops the tool adapter in Linux
tool_adapter/{adapter}/bin/	uninstall-{adapter}.sh	Cleans the tool adapter installation in Linux
tool_adapter/{adapter}/bin/	uninstall-{adapter}.bat	Removes the tool adapter from the Windows services register
tool_adapter/{adapter}/conf/	app.properties	Runtime configuration file
tool_adapter/{adapter}/conf/	{adapter}Configuration.xhtml	Configuration template. Several configuration templates may appear here.
tool_adapter/{adapter}/conf/db/	fill_Internal_Registry.sql	File to facilitate the registration of external tools in the internal tool registry
tool_adapter/{adapter}/lib/	{adapter}.jar	Tool adapter binary files
tool_adapter/{adapter}/lib/	*.jar	Other specific libraries
tool_adapter/{adapter}/log/	*.log	Log files
tool_adapter/{adapter}	COPYING.txt	GNU GPL license
tool_adapter/{adapter}	INSTALL.txt	Installation manual
tool_adapter/{adapter}	README.txt	Tool adapter license and third-party notices

4. In Linux, grant execution permissions to the `*.sh` files in `$GLUE_HOME/tool_adapters/{adapter}/bin` (`chmod +x $GLUE_HOME/tool_adapters/{adapter}/bin/*.sh`).

5. In Linux, run `$GLUE_HOME/tool_adapters/{adapter}/bin/install-{adapter}.sh`. The execution of that file: creates the file to provide persistence to the instances created with this adapter in `/srv/glue/tool_adapters/{adapter}/instances.txt`, and a symbolic link in `$GLUE_HOME/tool_adapters/{adapter}/data` pointing at this file; creates the empty log file `/var/log/glue/{adapter}.log`, and a symbolic link pointing at it in `$GLUE_HOME/tool_adapters/{adapter}/log`; and creates symbolic links at `/usr/bin` pointing at `$GLUE_HOME/tool_adapters/{adapter}/bin/start-{adapter}.sh` and `$GLUE_HOME/tool_adapters/{adapter}/bin/stop-{adapter}.sh`. In Windows, run `$GLUE_HOME/tool_adapters/{adapter}/bin/install-{adapter}.bat`. The execution of that file: creates the file `$GLUE_HOME/tool_adapters/{adapter}/data/instances.txt` to provide persistence to the instances created with this adapter; registers this adapter as a Windows service; and defines `$GLUE_HOME/tool_adapters/{adapter}/log/manager.log` as the log file (it is created the first time the adapter is started). By default, tool adapters are registered to be started at system boot time. The parameter *manual* should be passed to `install-{adapter}.bat` in order to force the manual start of the adapter.

Information for the internal tool registry

Every tool adapter must be registered in the internal tool registry to be invoked by the GLUElet Manager. In order to facilitate the registration process, tool adapters include the file `$GLUE_HOME/tool_adapters/{adapter}/conf/db/fill_Internal_Registry.sql`. Nevertheless, cautions must be taken on this file, since it is just a template. Therefore, several variables marked with square brackets must be replaced with suitable values according to the particular environment in which the tool adapter is deployed. After these modifications are made, the file can be executed in the SQL compliant database management console where the internal tool registry is located, in order to register the tools this adapter wraps. Significantly, if tool adapters are installed as part of the full GLUE! distribution package, rather than from individual packages, then, the `$GLUE_HOME/manager/conf/db/fill_Internal_Registry.sql` file may be used instead, in order to register all the tool adapters at once, as further explained in the `INSTALL.txt` file at `$GLUE_HOME`.

The first column in all the tables set by `fill_Internal_Registry.sql` is an integer that works as a primary key for every new row added. A value not used in previous rows must be selected for this integer. A careful check is advised, since all the identifiers but `[TOOL_IMP_ID_x]` are repeated in the `fill_Internal_Registry.sql` file as references in other tables apart from

those where they are primary keys. Therefore, the value of each identifier must be consistent throughout the file. Apart from these identifiers, some other values must be set in two tables:

- `InternalRegistry.ImplementationAdapters`. This is the URL where the tool adapter listens for requests. Significantly, the suffix `/ToolAdapter/{adapter}` cannot be changed. If several tool adapters and the GLUElet Manager are installed in the same host, then an example for the full URL in one of the tool adapters might be: `http://localhost:8186/ToolAdapter/{adapter}`. It is noteworthy that the port (`8186`) must be different for each tool adapter.
- `InternalRegistry.ToolImplementations`. This is the field in which the specific parameters required by each tool must be set. These parameters must be represented as a string containing the list of values for the parameters, following the URL encoding format. For instance, if institutional credentials are used for the management of external tool instances, these credentials must be set here.

Operation

Before starting a tool adapter, some runtime properties may be modified in its corresponding `$GLUE_HOME/tool_adapter/{adapter}/conf/app.properties` file:

- *port*. This is the number of the port where the tool adapter listens for requests from the GLUElet Manager.
- *logging*. This parameter must be *"on"* to save information in the log file about each request processed by the tool adapter.

Besides, to manually start and stop the tool adapters, it is recommended to follow these instructions:

- In Linux, use the commands `start-{adapter}` and `stop-{adapter}`.
- In Windows, access to the service management panel ("*Start*" button, "*Control Panel*", "*System and Maintenance*", "*Administration Tools*", "*Services*"), search and select the service with the name "*{Adapter}*" in the service list, and click the "*Init*" or "*Stop*" links at the top left corner of the list.

After that, the content of the `$GLUE_HOME/tool_adapter/{adapter}/log/{adapter}.log` can be checked to see if the tool adapter started without problems. In that case, a message like this should appear:

```
24-oct-2011 19:58:02 org.restlet.engine.http.connector.HttpServerHelper
start INFO: Starting the default [HTTP/1.1] server on port 8186
```

Final notes

Some final notes must be considered when installing and configuring tool adapters:

1. If the access to the tool adapter must be restricted to a limited set of system users for its start/stop, then it is important to keep in mind that authorized users need: execution permission on `$GLUE_HOME/tool_adapter/{adapter}/bin/*.sh`; read permissions on `$GLUE_HOME/tool_adapter/{adapter}/conf/app.properties`, and all the *jar* files; and write permissions on `/var/log/glue/{adapter}.log`.

2. In Windows systems, every time an adapter is started, its log file is erased.

D.3 Installation and configuration of VLE adapters

The Moodle adapter

The Moodle adapter, which has been developed for the Moodle 1.9 version, is installed as any other Moodle module[8]. Once installed, the Moodle administrator must configure the Moodle adapter, so this adapter can send requests to the GLUElet Manager. In order to configure the Moodle adapter, the Moodle administrator must go to "*Notifications*", "*Modules*", "*Activities*", "*GLUElet*". Two parameters must be set in this screen: the URL of the GLUElet Manager; and a timeout. This timeout prevents the Moodle adapter from hanging up if the connection with the GLUElet Manager or with the external tools cannot be successfully accomplished. Figure 4 depicts a screenshot of Moodle showing examples for these two parameters.

The LAMS adapter

The LAMS adapter, which has been developed for the LAMS 2.2 version, is installed as any other LAMS add-on[9]. Once installed, the LAMS administrator must configure the LAMS adapter, so this adapter can establish a communication with the GLUElet Manager. In order to configure the LAMS adapter, the LAMS administrator must go to "*Sys Admin*", "*Tool Management*", "*GLUElet*", "*Tool Management*", and set the URL of the GLUElet Manager, and the timeout for each request. Figure 5 depicts a screenshot of LAMS showing examples for these two parameters.

[8] `http://docs.moodle.org/en/Installing_contributed_modules_or_plugins`. Last visited: January 2012

[9] `http://wiki.lamsfoundation.org/display/lams/Downloads`. Last visited: June 2012.

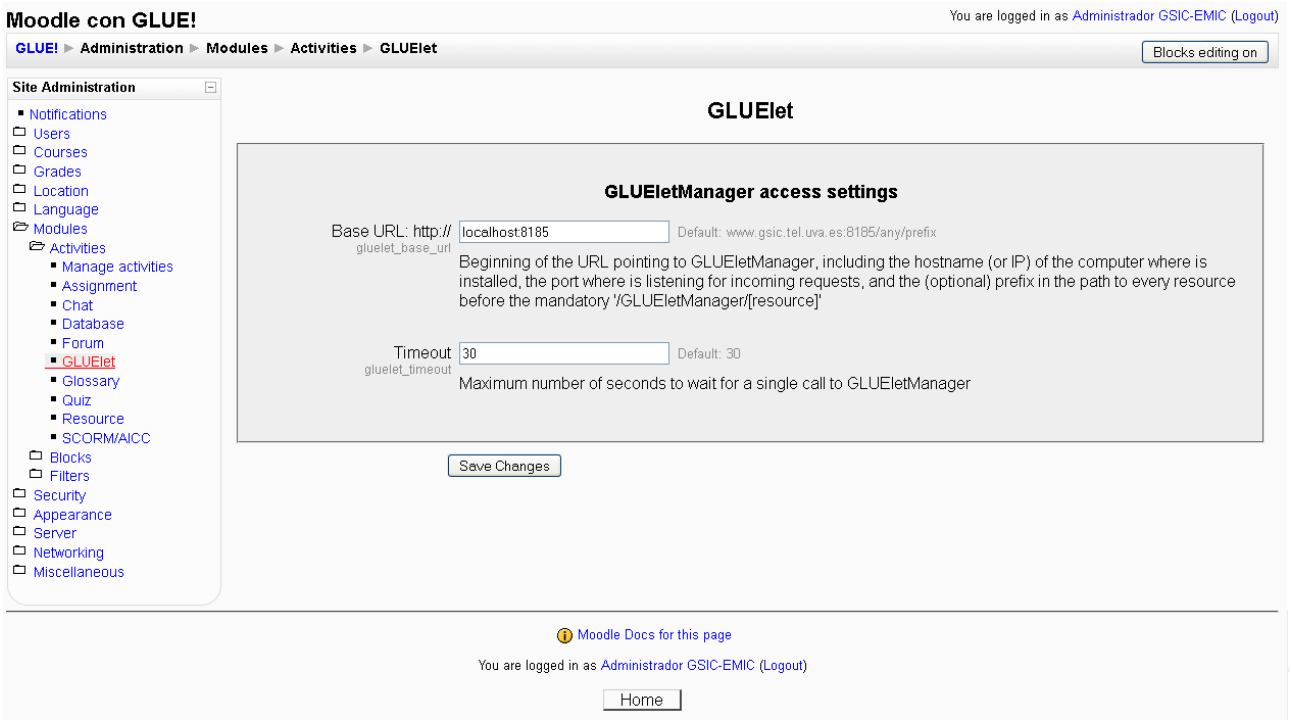

Figure 4: Moodle screenshot illustrating the configuration of the Moodle adapter. The Moodle administrator must specify the URL of the GLUElet Manager and a timeout in seconds to wait for every single call to the GLUElet Manager.

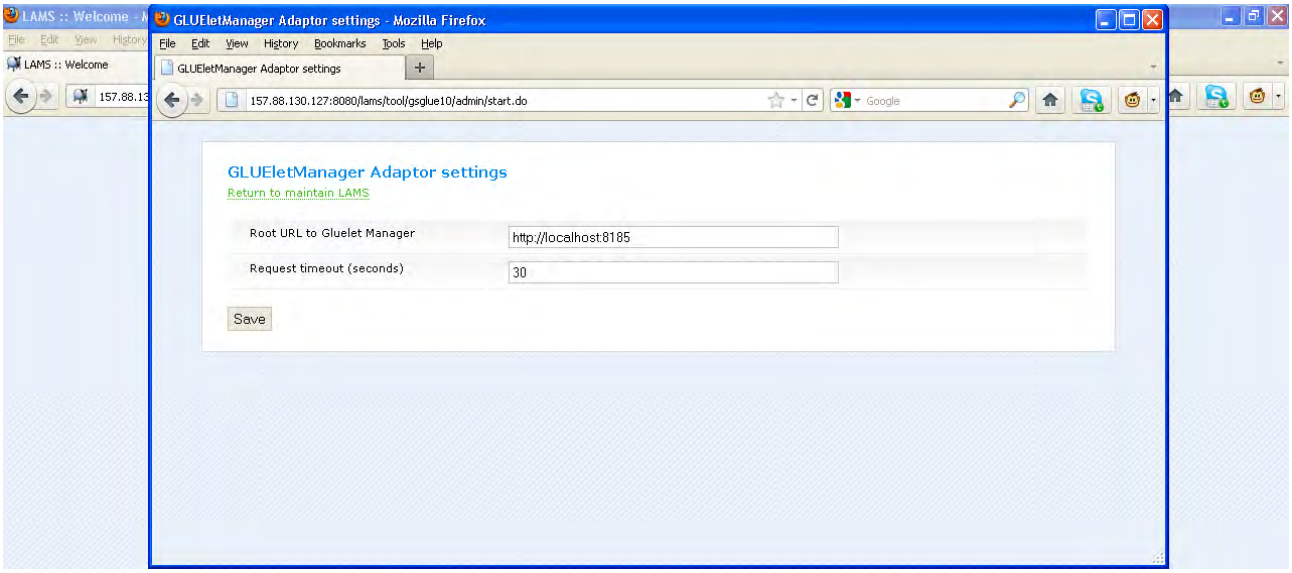

Figure 5: LAMS screenshot showing the configuration of the LAMS adapter. The LAMS administrator must determine the URL of the GLUElet Manager and a timeout in seconds to wait for every single call to the GLUElet Manager.

The MediaWiki adapter

The MediaWiki adapter, which has been developed for the MediaWiki 1.16 version, is installed as any other MediaWiki extension[10]. MediaWiki administrators do not have a user interface in MediaWiki to configure this adapter, as it happens in other VLEs like Moodle or LAMS. Instead, they need to manually edit the `localsettings.php` file in their MediaWiki installation, and add the following code:

```
require_once("$IP/extensions/MediaWikiVLEGlueExtension/MediaWikiVLEGlueExtension.php");
$wgGlueExtensionDefaultUser = "GlueForMediaWikiDefaultUser";
$wgNewGlueExtensionGlueletManagerURL = "http://localhost:8185/GLUEletManager";
```

The first statement indicates the path of the main `.php` file in the MediaWiki adapter. The second sentence designates the default user in MediaWiki. The third sentence points at the URI of the GLUElet Manager.

[10] `http://mediawiki.org/wiki/Manual:Extensions`. Last visited: June 2012.

Appendix E: Examples of usage

The reference implementation of the GLUE! architecture, GLUE!-RI, has been introduced in chapter 4. GLUE!-RI includes the GLUE! core, nine tool adapter, and three VLE adapters for Moodle, LAMS and MediaWiki. Therefore, end-users can use GLUE!-RI for the integration of external tools in Moodle, LAMS, and MediaWiki. This appendix illustrates with some examples the usage for each of these environments.

E.1 User manual for Moodle

This manual is oriented to end-users that employ GLUE! for the integration of external tools in the VLE Moodle. Screenshots depict the steps for educators using Moodle, as well as for students participating in a Moodle course.

Educator view

Educators can integrate external tools in a Moodle course with GLUE! by managing the tool life cycle within this VLE interface. Figures 6 to 9 show the four steps that educators must follow in order to create and configure tool instances of Google Documents for the users and groups defined in a Moodle course. In this example, two groups (*group1* and *group2*) have previously been defined. Besides, *student1* joined *group1*, and *student2* joined *group2*.

Figure 6 shows the overview of the course, including a previously created "*GLUElet*" activity. In order to create new instances (*gluelets*), the educator must click on the "*Add an activity*" drop-down menu in the course homepage, and select the *GLUElet* option. Then, educators can configure the Moodle activity (as usual) in a new screen, indicating, for instance, a name and a description. Besides, in this screen a new drop-down menu appears ("*Tool*") with the list of available external tools. Figure 7 shows this screen, including the selection of Google Documents, and also the "*Group mode*": the option "*no groups*" creates one single external tool instance, which is shared among all the students registered in that course; the option "*separate groups*" creates one external tool instance for each group defined in the course, and students can only access the

instances of the group they belong to; the option "*visible groups*" creates one instance for each group defined in the course, although students can access their partners' instances too. This also happens with Moodle built-in tools.

After that, a new screen shows up allowing the configuration of each of the external tool instances that are going to be created. Besides, in this step, educators may also decide to reuse other instances of the same tool that were created earlier in the course. Figure 8 presents this screen, which is added by the Moodle adapter. In the case of Google Documents a title and an initial file can be set. Finally, Figures 9 to 12 show the visualization of different external tool instances embedded in Moodle, once created. Remarkably, the educator can modify the group mode or the members of each group at any time, and the architecture will automatically update the students that share each external tool instance.

Student view

After the educator creates the activities, which may include both built-in and external tools, students log in Moodle to enact a collaborative learning situation (e.g. Figure 13). Student can only see the tool instances shared with their group partners, unless the educator had selected "*visible groups*", when creating a Moodle activity.

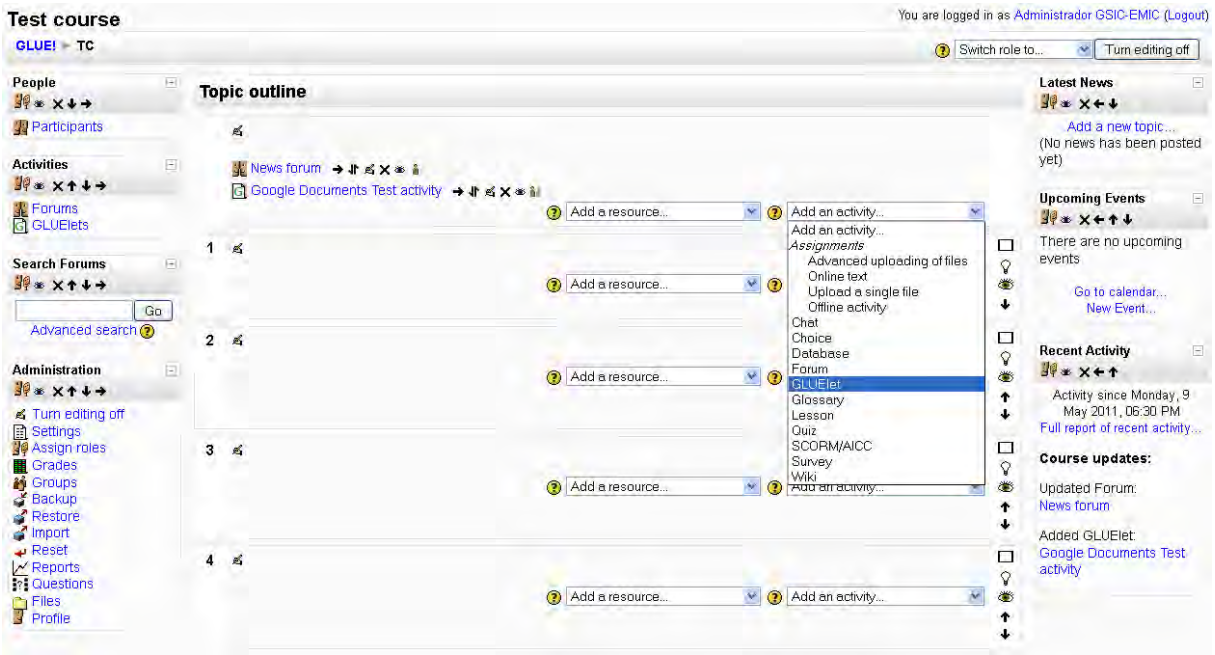

Figure 6: Moodle screenshot showing the creation of a new "*GLUElet*" activity. The educator can select a built-in tool or an external tool (*GLUElet* option in the drop-down activity menu).

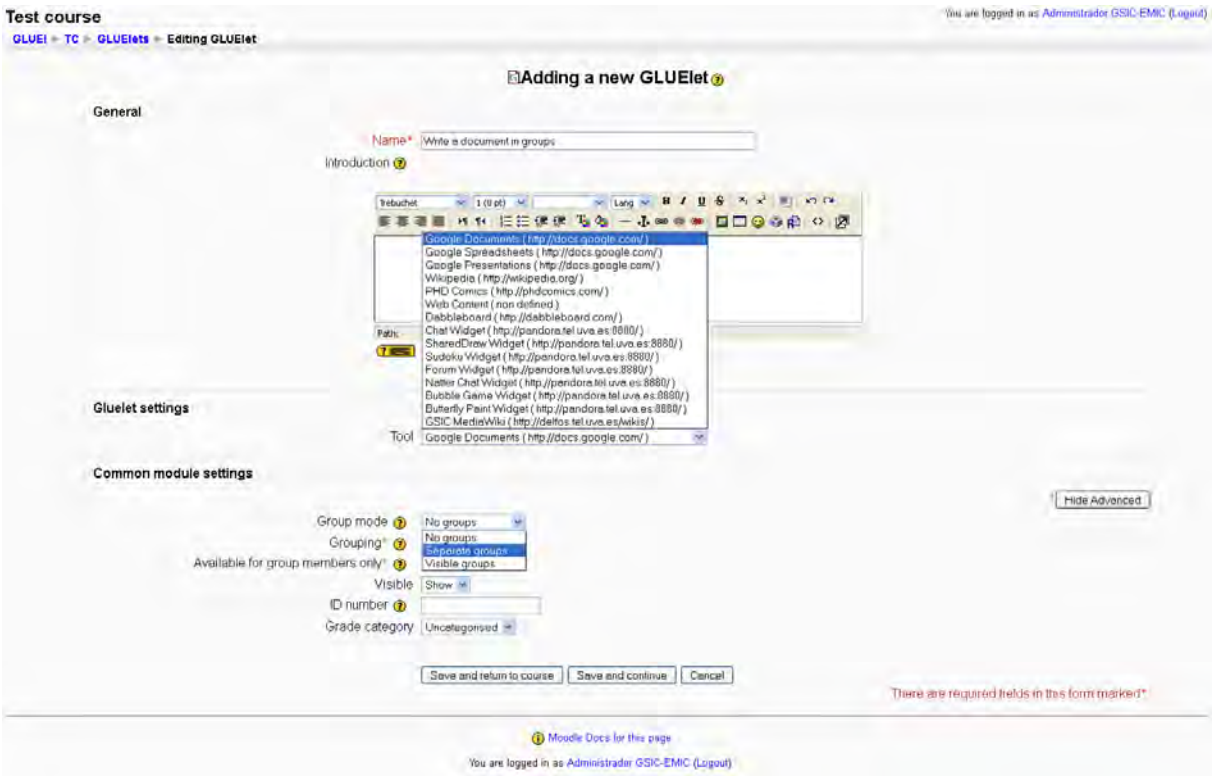

Figure 7: Moodle screenshot showing the configuration of the activity. Here, the educator selects the external tool (Google Documents), and the group mode ("*separate groups*") from two different drop-down menus.

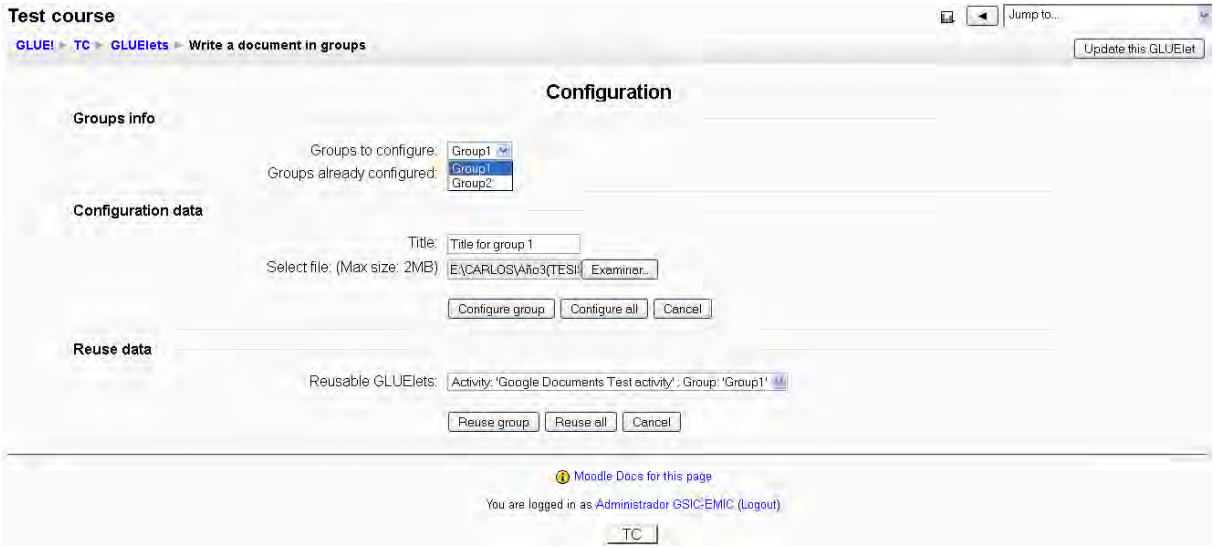

Figure 8: Moodle screenshot showing the configuration of external tool instances. The educator can provide an initial configuration for each instance (group). In this case, a title and an initial file can be set for Google Documents instances. The educator can also reuse existing instances of this tool in the same Moodle course.

Figure 9: Moodle screenshot showing the visualization of tool instances for Google Documents. These instances are embedded in the Moodle interface and have been configured with different titles and initial documents for each group.

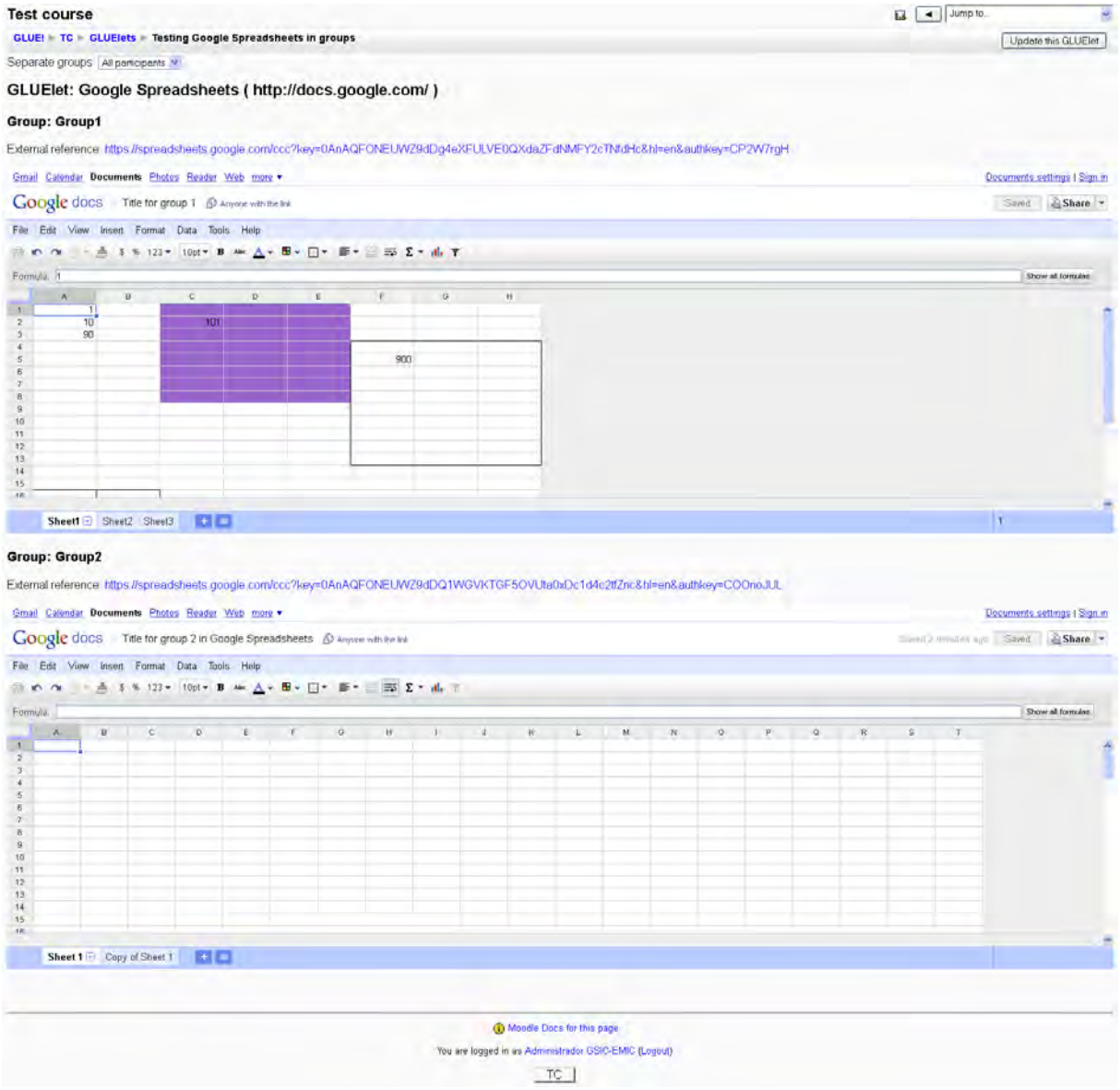

Figure 10: Moodle screenshot showing the visualization of tool instances for Google Spreadsheets. Tool instance for *group1* has been configured with a title ("*Title for group 1*") and an initial file. Tool instance for *group2* has been configured with a different title ("*Title for group 2 in Google Spreadsheets*") but it shows a blank spreadsheet, since no initial file was provided.

Figure 11: Moodle screenshot showing the visualization of tool instances for the Natter Chat widget. These instances have not been configured during the instantiation of the corresponding activity. Nevertheless, instance for *group1* shows a message that the Moodle administrator (playing the role of the educator in this course) left to the students belonging to this group. Instance for *group2* shows the list of users that are sharing it: *student2* and *admin*.

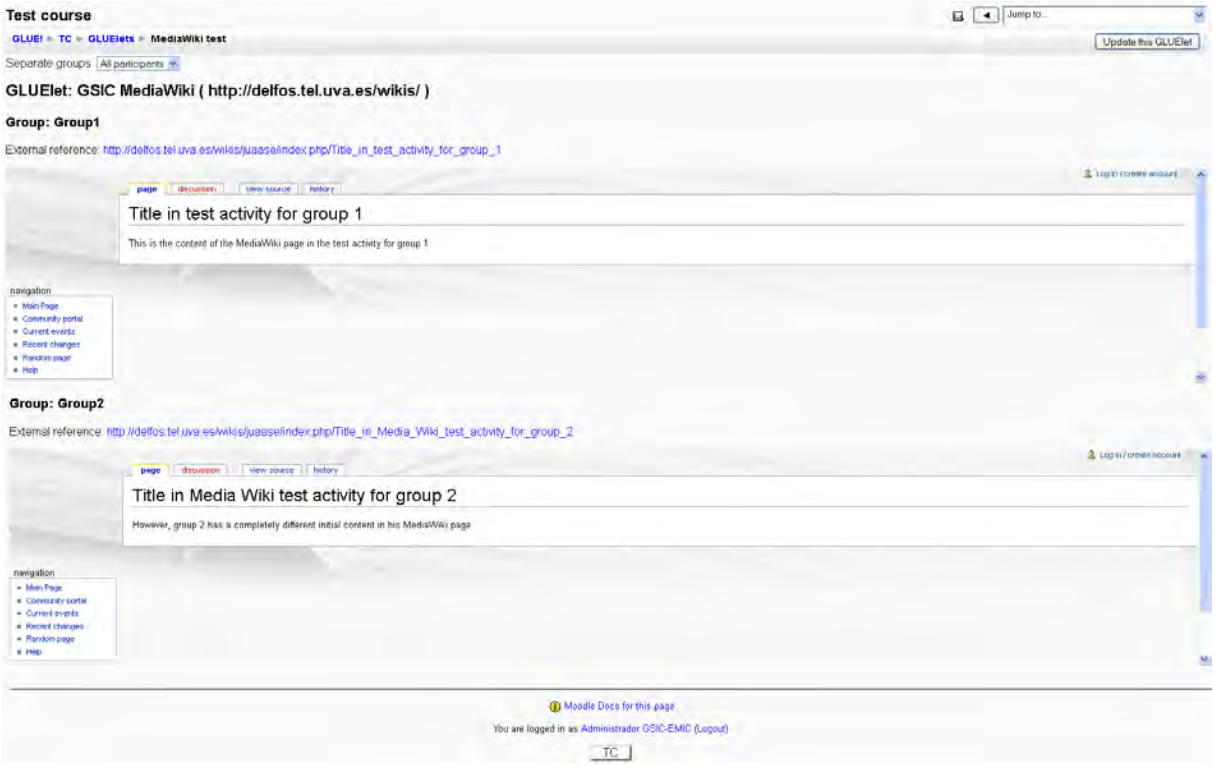

Figure 12: Moodle screenshot showing the visualization of tool instances for MediaWiki. These instances have been configured with different titles and initial contents for each group.

Figure 13: Moodle screenshot showing the visualization of a Google Document instance for a member of *group1* (*student1*). The option "*separate groups*" was chosen when creating the instances for this activity.

E.2 User manual for LAMS

This manual is oriented to end-users that use GLUE! for the integration of external tools in LAMS. Screenshots depict the steps for authors designing lessons in LAMS, monitors deploying lessons in LAMS courses, and students (learners) participating in the enactment of LAMS lessons.

Author view

Authors can add external tools in a LAMS lesson with GLUE! in the authoring environment. Figures 14 to 16 show the steps that authors must follow to add Google Documents in a learning sequence. Figure 14 shows the set of LAMS tools on the left side, including the "*Gluelet*" activity. This screenshot also shows a sequence of activities with four external tools and two different group settings. Significantly, the use of LAMS pathways, branching and stop signs is also supported when adding external tools. Once the "*Gluelet*" activity is added to the sequence, it can be configured as any other LAMS built-in tool (see Figure 15). That configuration includes basic parameters, like the name or the description of the activity, but also a drop-down menu to select the external tool. Once selected, the configuration template is rendered in the same screen, to be filled out by the author. Figure 16 shows the result of configuring a *Gluelet* activity that adds the Google Documents tool to a LAMS lesson. When everything is set, the lesson can be saved and deployed in the monitoring environment. Furthermore, this lesson can be exported to be reused by other authors using LAMS. It is noteworthy that the authoring environment enables the creation of abstract designs, which must be later particularized in the monitoring environment, considering the actual learners registered in the LAMS course.

Monitor view

Monitors can deploy LAMS lessons that include external tools in their courses as usual. External tool instances are created after the groups defined in the authoring environment are populated, and just at the moment a student first access an activity. Monitors can modify group members in those activities that have not started, and even add more students to a group in those activities that are running. In both cases, GLUE! automatically updates the students that share each external tool instance.

Learner view

Once learners log in LAMS, they can realize the learning sequence using both built-in tools and external tools. Each learner can only see the tool instances intended for his group in each activity. Examples for Dabbleboard and Google Documents are shown in Figures 17 and 18.

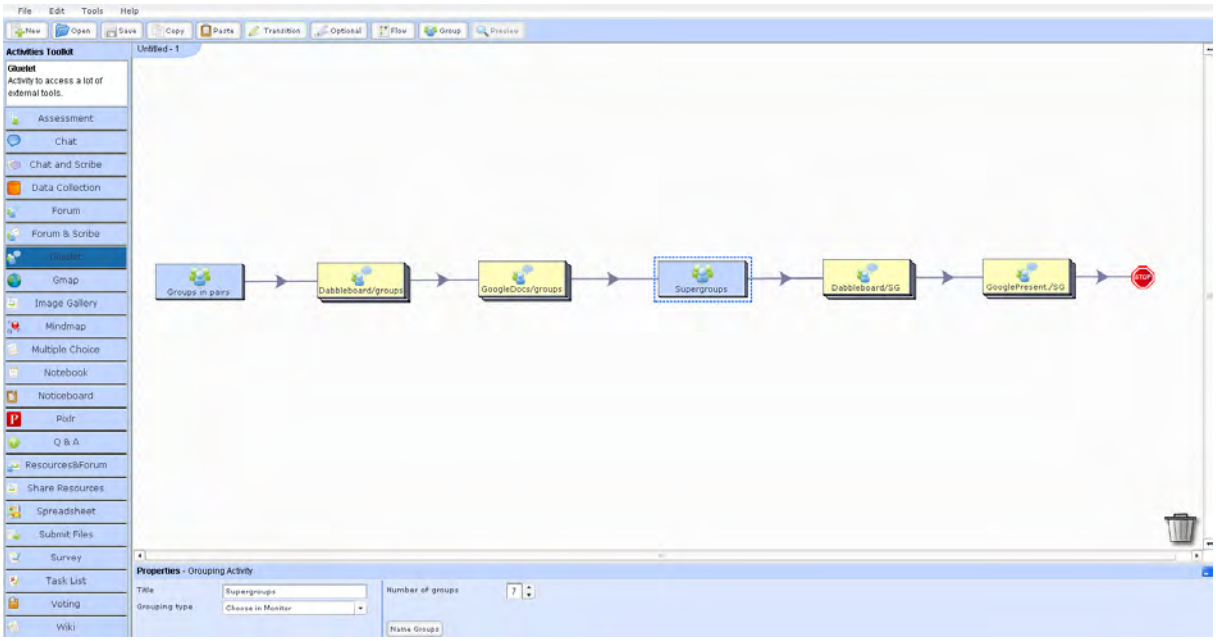

Figure 14: LAMS screenshot showing a sequence of activities that includes four external tools ("*Gluelet*" activities) and two group settings.

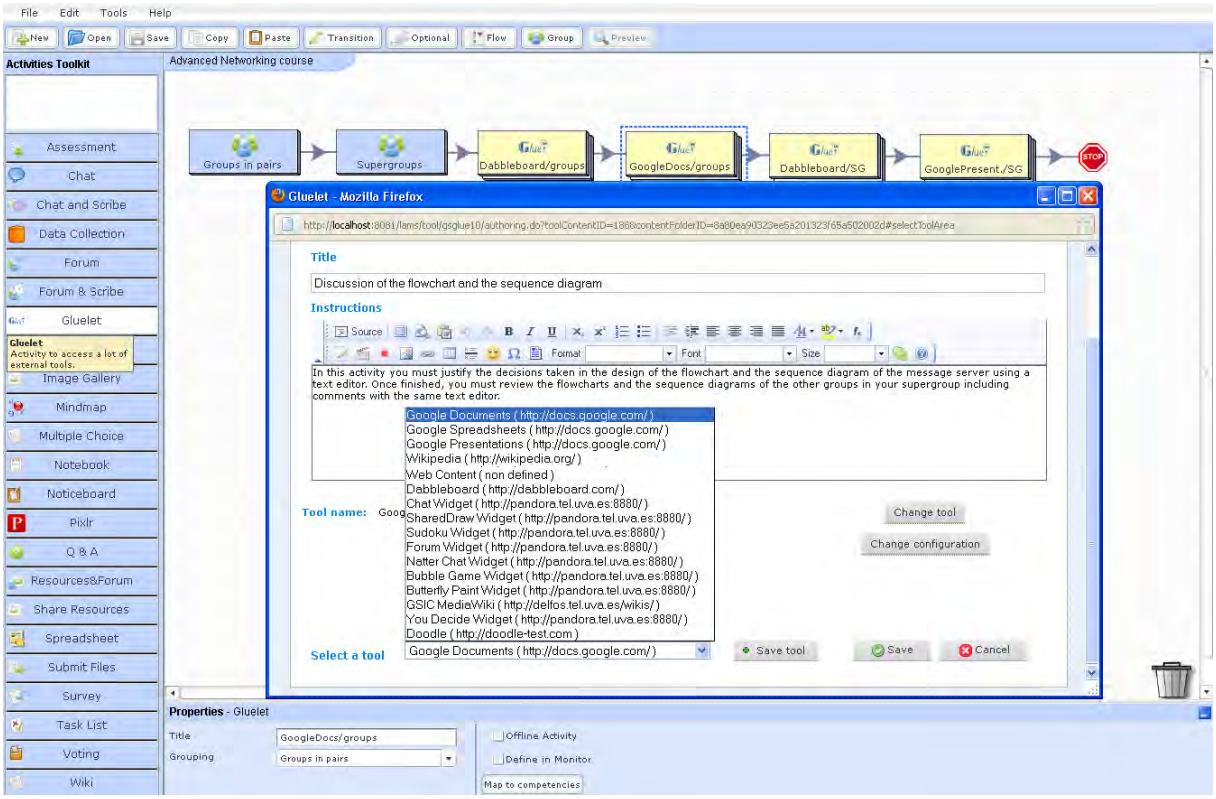

Figure 15: LAMS screenshot showing the configuration of a "*Gluelet*" activity. Apart from the basic settings, like the name or the description of the activity, an external tool can be selected and later configured.

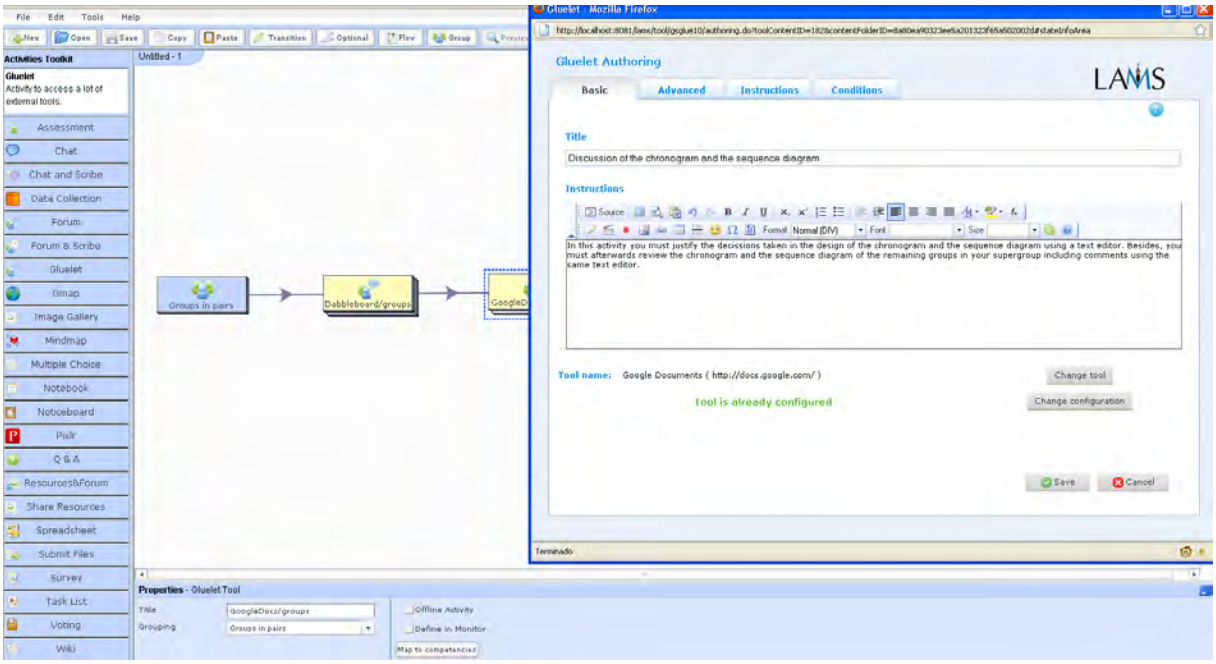

Figure 16: LAMS screenshot showing the final settings of a "*Gluelet*" activity that adds Google Document to the learning design.

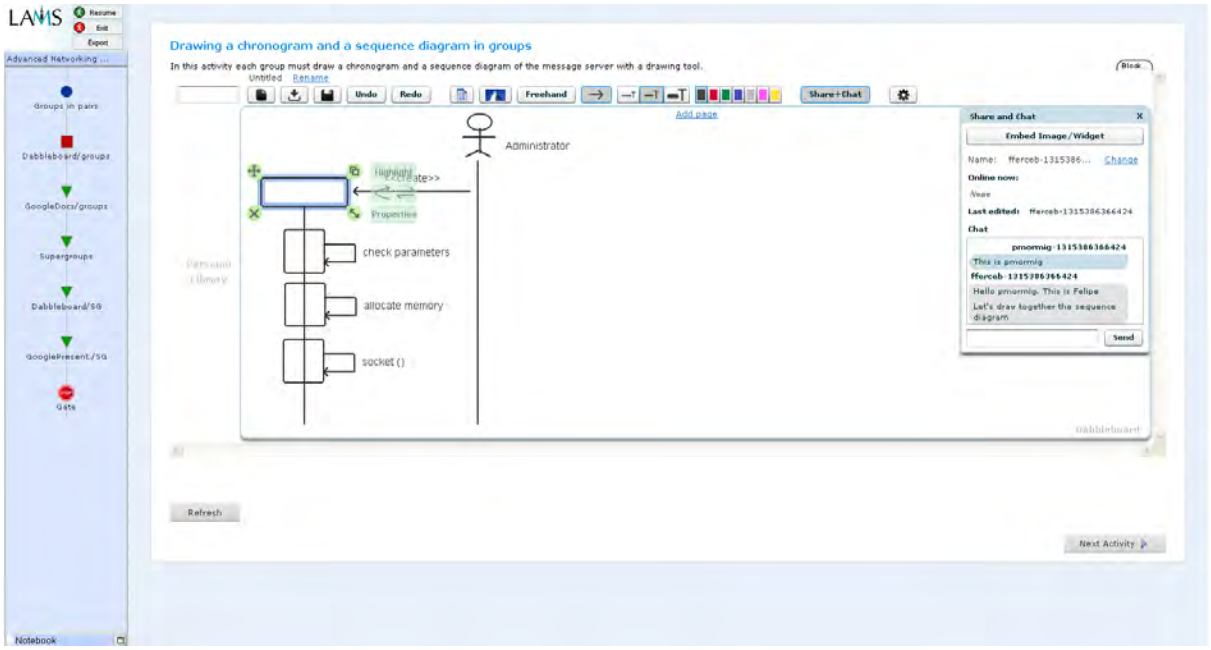

Figure 17: LAMS screenshot showing the visualization of a Dabbleboard tool instance.

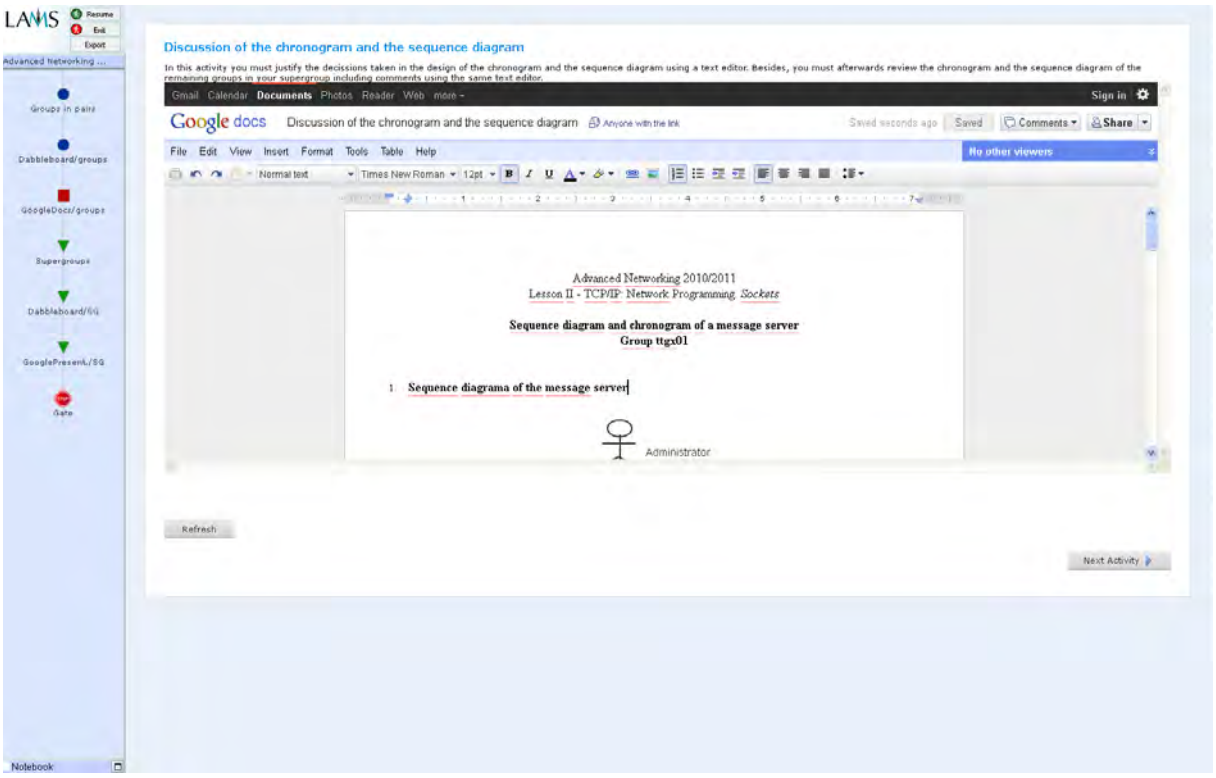

Figure 18: LAMS screenshot showing the visualization of a Google Documents tool instance.

E.3 User manual for MediaWiki

This manual is oriented to end-users that use GLUE! for the integration of external tools in MediaWiki. MediaWiki does not support the definition of different roles for educators and students, and so, screenshots depict the steps to integrate external tools for any end-user that can edit the content in a MediaWiki installation.

User view

The MediaWiki adapter allows any user that can edit a MediaWiki page to manage the life cycle of external tools within the MediaWiki interface. External tool instances are added to a MediaWiki page associated to `<gluelet>` tags. These instances (and tags) can be copied and pasted to any other page in the same MediaWiki installation, in order to reuse them in different contexts or activities. Nevertheless, users should be careful when deleting instances, since they may be on use in several pages.

MediaWiki is not a VLE, and so, it does not support the concepts of *course*, *activity* or *group*. However, different courses and activities can be created in a hierarchical structure of MediaWiki pages. Nevertheless, any user that can access a MediaWiki content, would be able to access any tool instance, insofar as different group configurations and permissions are not supported.

Figures 19 to 26 illustrate the steps that end-users must follow to create and configure tool instances of Google Documents in a MediaWiki page. Figure 19 shows the login of a registered user in MediaWiki. Figure 20 shows the edition of a MediaWiki page. Here, users can create tool instances by clicking on "*new gluelet*". They can also edit the content of the MediaWiki page as usual, adding text, images, hyperlinks, etc. If the user decides to add a new *gluelet*, a window for the selection of the external tool (e.g. Google Documents) appears, as it is shown in Figure 21. Figure 22 shows the configuration allowed for this tool. In the case of Google Documents, the title and an initial file can be set. Figure 23 shows a message confirming the creation of the Google Documents instance and an optional field to name the instance in the MediaWiki page (this is a local name shown when visualizing the instance). Figure 24 shows the `<gluelet>` tag (including a *glueletId* for this tool instance and the name previously assigned). This tag is automatically generated by the MediaWiki adapter. Besides, a message has been manually added after the `<gluelet>` tag. Figure 25 shows the visualization of the MediaWiki page, which includes the instance of Google Documents. Figure 26 shows the edition of the same page. Once the page is saved, tool instances can be deleted by clicking on their correspondent icon on the top left. Besides, the content of an instance (e.g. `<gluelet>/instance/210/</gluelet>`) can be copied and pasted to any other page in the same Mediawiki installation.

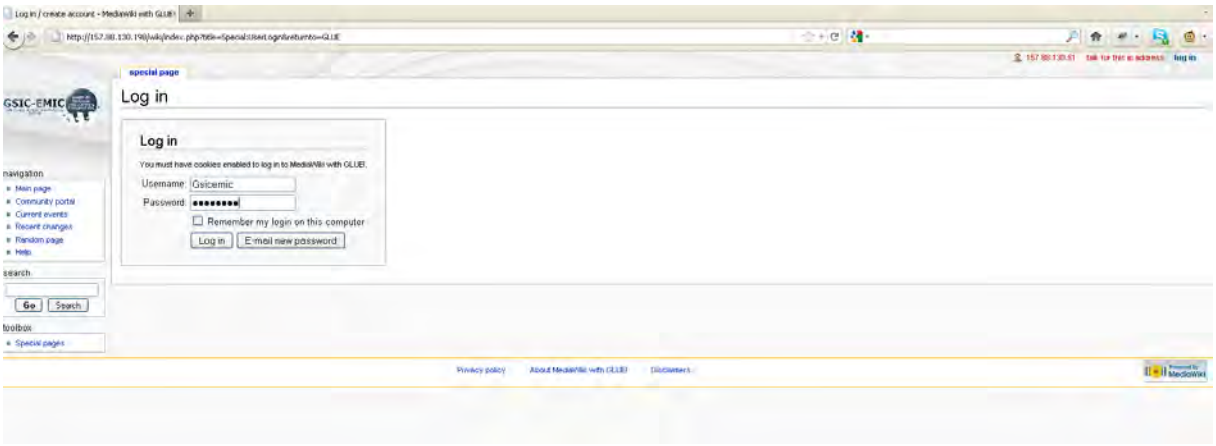

Figure 19: MediaWiki screenshot showing the authentication of a user in this platform.

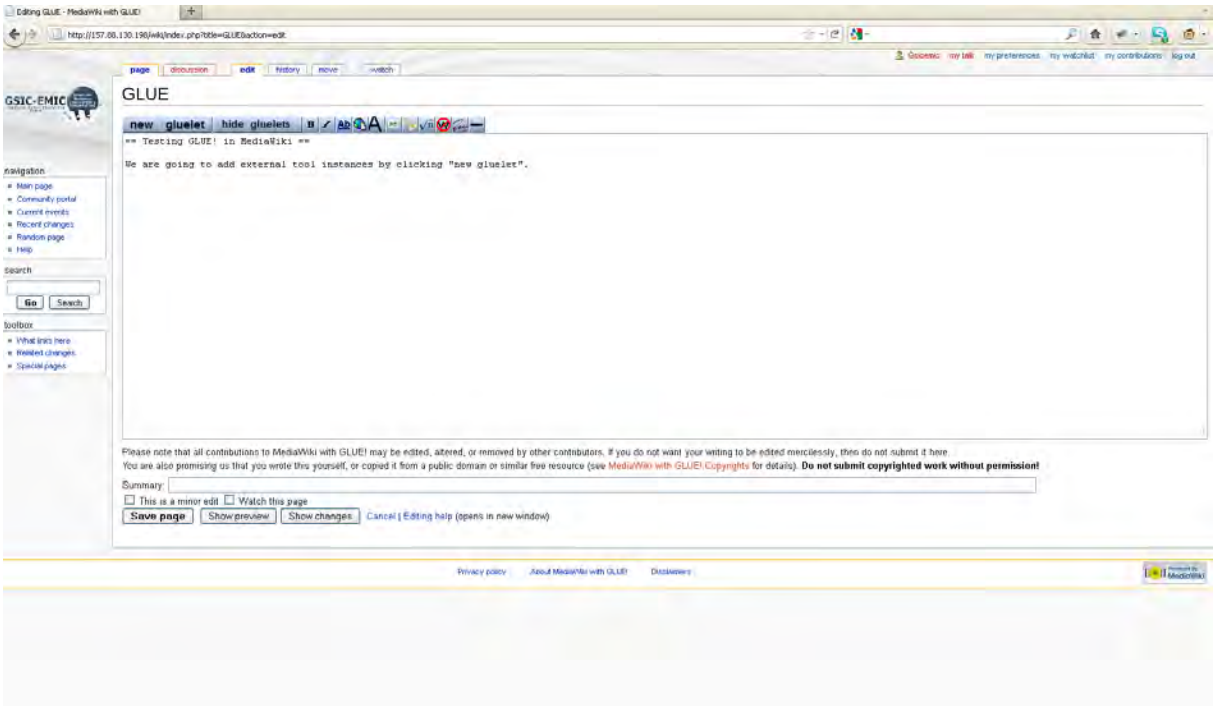

Figure 20: MediaWiki screenshot showing the edition of a MediaWiki page. Two buttons are added by the MediaWiki adapter. The "*new gluelet*" button enables end-users to create and configure external tool instances. The "*hide gluelets*" button enables users to hide the list of created gluelets in this page.

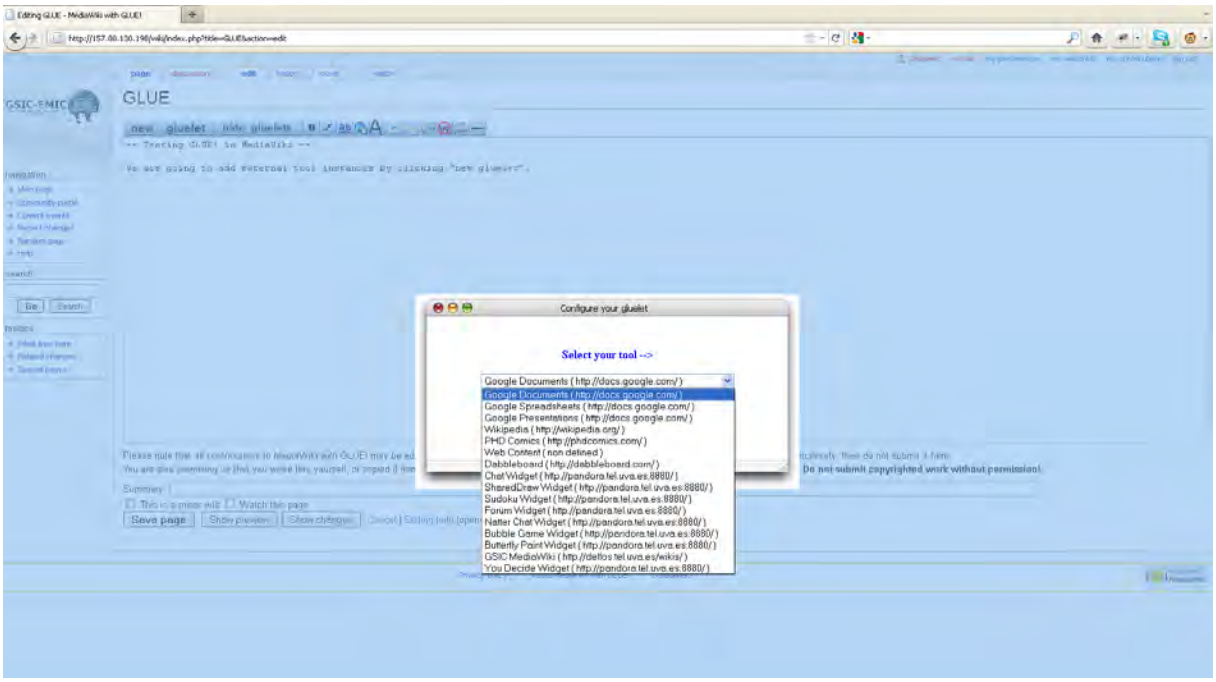

Figure 21: MediaWiki screenshot showing the creation of external tool instances. The first step is the selection of the external tool from a drop-down menu.

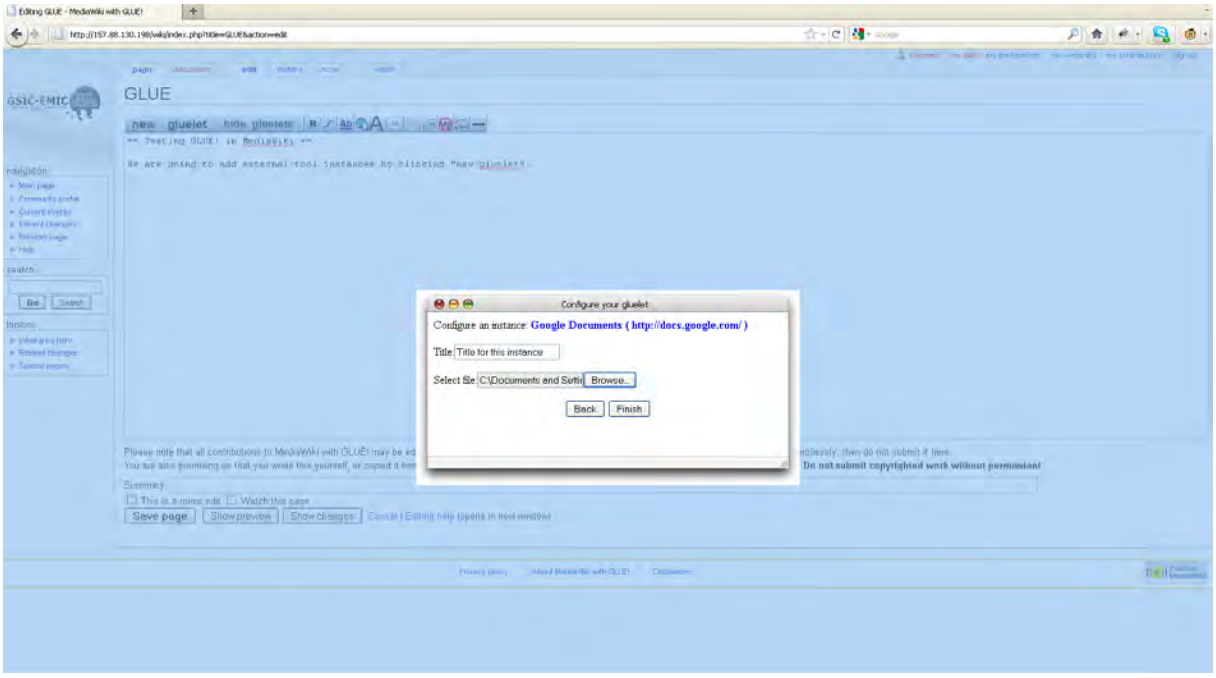

Figure 22: MediaWiki screenshot showing the configuration of a new Google Document. A title and an initial file can be set for this tool.

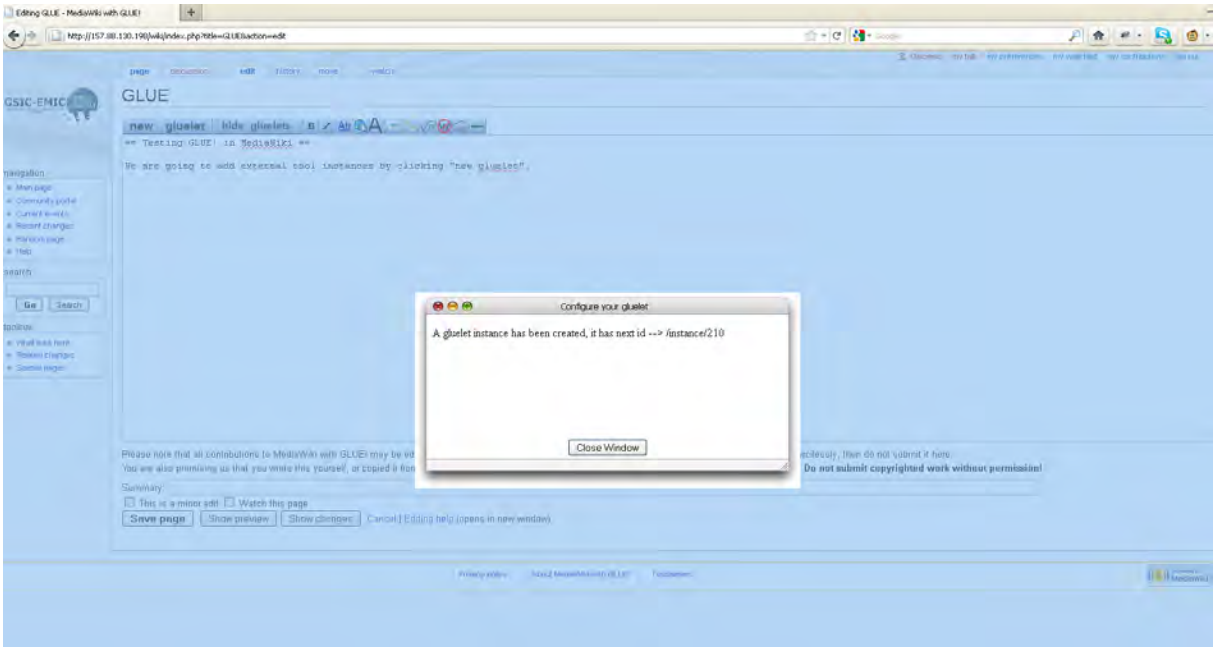

Figure 23: MediaWiki screenshot showing a message reporting that the external tool instance has successfully been created.

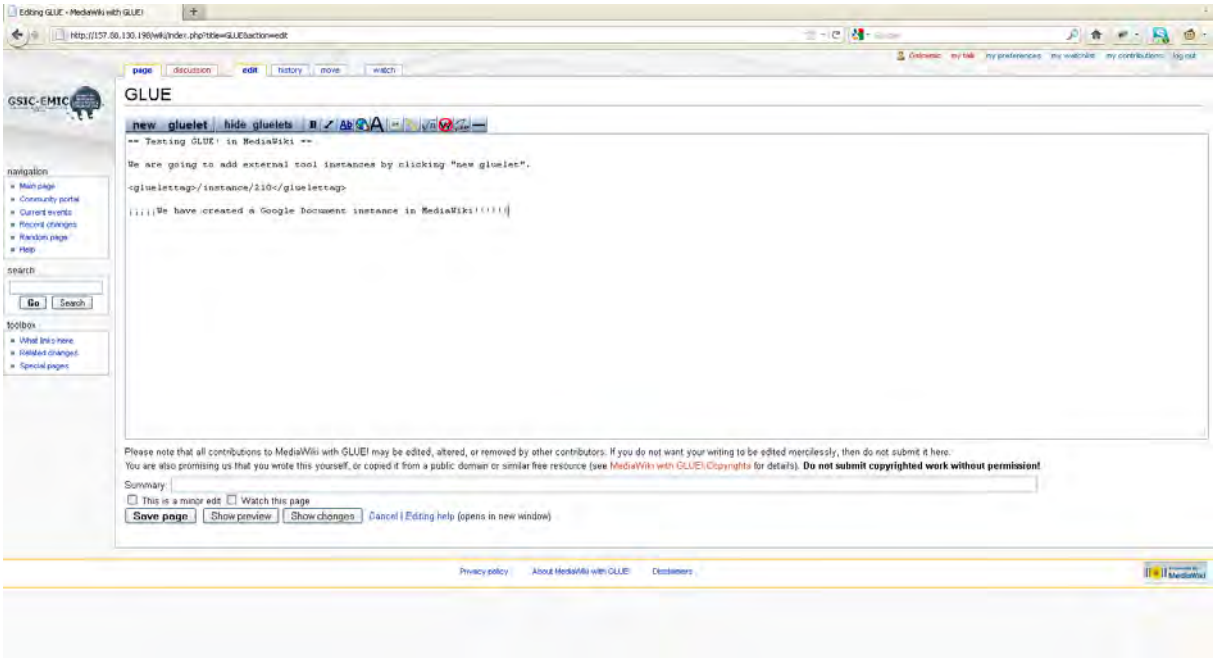

Figure 24: MediaWiki screenshot showing the edition of a MediaWiki page after the creation of an external tool instance. The `<gluelet>` tag includes a reference to this instance.

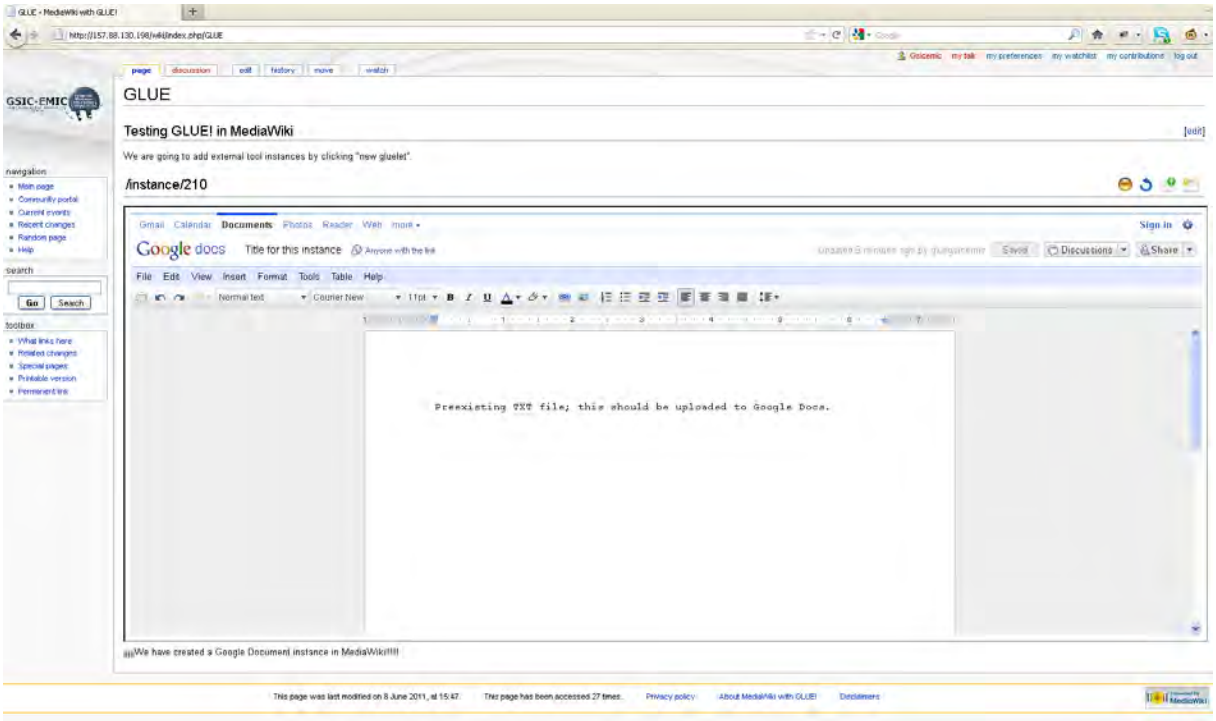

Figure 25: MediaWiki screenshot showing the visualization of the MediaWiki page including a Google Document.

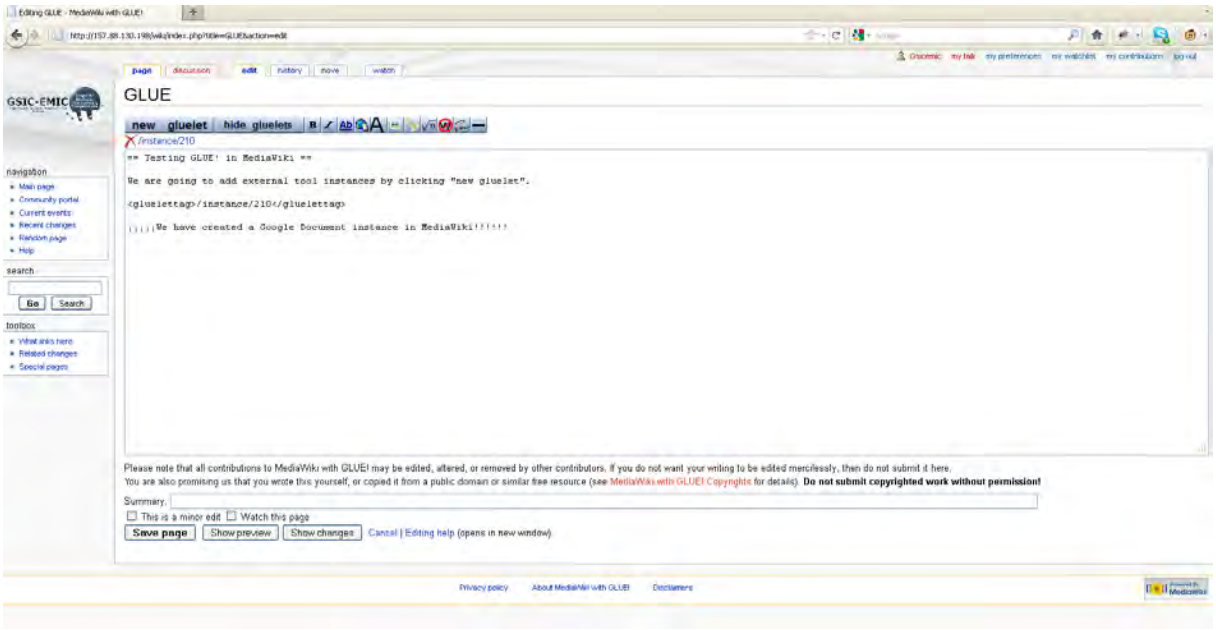

Figure 26: MediaWiki screenshot showing the edition of the same page, after it has been saved. A deletion icon appears on the top left for each tool instance that has been created.

References

[Abb01] C. Abbott. *ICT: Changing Education.* Routledge Falmer, London, UK, 2001.

[ADL04] ADL, Advanced Distributed Learning. SCORM 2004 4th Edition, 2009. URL: http://adlnet.gov/capabilities/scorm/scorm-2004-4th, last visited: June 2012.

[Adr93] W.R. Adrion. Research Methodology in Software Engineering: Summary of the Dagstuhl Workshop on Future Directions in Software Engineering. *ACM SIGSOFT Software Engineering Notes*, 18(1):36–37, 1993.

[Ajj08] H. Ajjan and R. Hartshorne. Investigating faculty decisions to adopt Web 2.0 technologies: Theory and Empirical Tests. *The Internet and Higher Education*, 11(2):71–80, 2008.

[Ala09] C. Alario-Hoyos, E. Gómez-Sánchez, M. Bote-Lorenzo, G. Vega-Gorgojo, and J.I. Asensio-Pérez. Grid Service-Based Benchmarking Tool for Computer Architecture Courses. In *Proceedings of the 4th European Conference on Technology Enhanced Learning, ECTEL 2009*, pages 621–626, Nice, France, September-October 2009. Springer-Verlag, LNCS 5794.

[Ala10a] C. Alario-Hoyos, J.I. Asensio-Pérez, M.L. Bote-Lorenzo, E. Gómez Sánchez, G. Vega-Gorgojo, and A. Ruiz-Calleja. Integration of external tools in Virtual Learning Environments: main design issues and alternatives. In *Proceedings of the 10th IEEE International Conference on Advanced Learning Technologies, ICALT 2010*, pages 384–388, Sousse, Tunisia, July 2010. IEEE Computer Society.

[Ala10b] C. Alario-Hoyos, E. Gómez-Sánchez, M.L. Bote-Lorenzo, J.I. Asensio-Pérez, A. Ruiz-Calleja, and G. Vega-Gorgojo. Towards single sign-on in the integration of external tools in Virtual Learning Environments. In *Proceedings of the 9th Workshop on Telematic Engineering, JITEL 2010*, pages 285–288, Valladolid, Spain, September 2010. In Spanish.

[Ala10c] C. Alario-Hoyos and S. Wilson. Comparison of the main alternatives to the integration of external tools in different platforms. In *Proceedings of the International Conference of Education, Research and Innovation, ICERI 2010*, pages 3466–3476, Madrid, Spain, November 2010. IATED.

[Ala11a] C. Alario-Hoyos, O. García-García, and E. Gómez-Sánchez. Problem-based learning supported by web platforms and tools in a Software Engineering course. In *Proceedings of the 2nd Workshop on Educational Innovation, JIE 2011*, pages 26–33, Santander, Spain, September 2011. In Spanish.

[Ala11b] C. Alario-Hoyos, M.L. Bote-Lorenzo, E. Gómez-Sánchez, D.A. Velasco-Villanueva, J.I. Asensio-Pérez, G. Vega-Gorgojo, and A. Ruiz-Calleja. Integration of external tools with GLUE! in LAMS: requirements, implementation and a case study. In *Proceedings of the 6th International LAMS and Learning Design Conference, LAMS 2011*, pages 40–48, Sydney, Australia, December 2011.

[Ala12a] C. Alario-Hoyos, M.L. Bote-Lorenzo, E. Gómez-Sánchez, J.I. Asensio-Pérez, G. Vega-Gorgojo, and A. Ruiz-Calleja. GLUE!: An Architecture for the Integration of External Tools in Virtual Learning Environments. *Computers & Education (submitted)*, 2012.

[Ala12b] C. Alario-Hoyos, M.L. Bote-Lorenzo, E. Gómez-Sánchez, J.I. Asensio-Pérez, G. Vega-Gorgojo, and A. Ruiz-Calleja. Integration of external tools in VLEs with the GLUE! architecture: A case study. In *Proceedings of the 7th European Conference on Technology Enhanced Learning, ECTEL 2012 (submitted)*, 2012.

[Ala12c] C. Alario-Hoyos, M.L. Bote-Lorenzo, E. Gómez-Sánchez, J.I. Asensio-Pérez, G. Vega-Gorgojo, and A. Ruiz-Calleja. Demonstration of the integration of external tools in VLEs with the GLUE! architecture. In *Proceedings of the 7th European Conference on Technology Enhanced Learning, ECTEL 2012 (submitted)*, 2012.

[Ala12d] C. Alario-Hoyos, J.A. Muñoz-Critóbal, L.P. Prieto, M.L. Bote-Lorenzo, J.I. Asensio-Pérez, and E. Gómez-Sánchez. GLUE! - GLUE!-PS: An approach to deploy non-trivial collaborative learning situations that require the integration of external tools in VLEs. In *Proceedings of the 1st Moodle Research Conference (submitted)*, 2012.

[Alb83] A.J. Albretch and J.E. Gaffney. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering*, 9(6):639–648, 1983.

[App99] W. Appelt and P. Mambrey. Experiences with the BSCW Shared Workspace System as the Backbone of a Virtual Learning Environment for Students. In *Proceedings of*

the World Conference on Educational Multimedia, Hypermedia and Telecommunications, ED-MEDIA 1999, pages 1710–1715, Seattle, WA, USA, June 1999. AACE.

[Arm05] V. Armstrong, S. Barnes, R. Sutherland, S. Curran, S. Mills, and I. Thompson. Collaborative research methodology for investigating teaching and learning: the use of interactive whiteboard technology. *Educational Review*, 57(4):455–469, 2005.

[Ase08] J.I. Asensio-Pérez, M.L. Bote-Lorenzo, G. Vega-Gorgojo, Y. Dimitriadis, E. Gómez-Sánchez, and E.D. Villasclaras-Fernández. Adding mash-up based tailorability to VLEs for scripted Collaborative Learning. In *Proceedings of the 1st International Workshop on Mashup Personal Learning Environments, MUPPLE 2008*, pages 14–17, Maastricht, The Netherlands, September 2008.

[Att07] G. Attwell. Personal Learning Environments - The Future of eLearning? *eLearning Papers*, 2(1):1–8, 2007. URL: http://elearningeuropa.info/en/download/file/fid/19297, last visited: June 2012.

[Aug04] N. Augar, R. Raitman, and W. Zhou. Teaching and learning online with wikis. In *Proceedings of the 21st Australasian Society for Computers in Learning in Tertiary Education, ASCILITE 2004*, pages 95–104, Perth, Australia, December 2004.

[Avo04] N. Avouris, M. Margaritis, and V. Komis. Modelling interaction during small-groups synchronous problem-solving activities: The Synergo approach. In *Proceedings of the the 2nd International Workshop on Designing Computational Models of Collaborative Learning Interaction*, pages 13–18, Maceió, Brazil, September 2004.

[Bak96] M. Baker and K. Lund. Flexibly structuring the interaction in a CSCL environment. In *Proceedings of the European Conference on Artificial Intelligence in Education, EuroAIED 1996*, pages 401–407, Lisbon, Portugal, September-October 1996.

[Bak97] M. Baker and K. Lund. Promoting reflective interactions in a CSCL environment. *Journal of Computer Assisted Learning*, 13(1):175–193, 1997.

[Bal09] N. Balacheff, S. Ludvigsen, T. de Jong, A. Lazonder, and S. Barnes (eds.). *Technology-Enhanced Learning: Principles and Products*. Springer, Berlin, Germany, 2009.

[Bat03] A.W. Bates and G. Poole. *Effective Teaching with Technology in Higher Education*. Jossey-Bass Publishers, San Francisco, CA, USA, 2003.

[Bav03] T. Baving, D. Cook, and T. Green. Integrating the Educational Enterprise. Technical report, Department of Computer Science, University of Cape Town, South Africa,

2003. URL: http://pubs.cs.uct.ac.za/archive/00000089/01/paper.pdf, last visited: June 2012.

[Ber01] A. Berger, R. Moretti, P. Chastonay, P. Dillenbourg, A. Bchir, R. Baddoura, C. Bengondo, D. Scherly, P. Ndumbe, P. Farah, and B. Kayser. Teaching Community Health By Exploiting International Socio-Cultural and Economical Differences. In *Proceedings of the 1st European Conference on Computer Supported Collaborative Learning, EuroCSCL 2001*, pages 97–105, Maastricht, The Netherlands, March 2001.

[Ber05] A. Berggren, D. Burgos, J.M. Fontana, D. Hinkelman, V. Hung, A. Hursh, and G. Tielemans. Practical and Pedagogical Issues for Teacher Adoption of IMS Learning Design Standards in Moodle LMS. *Journal of Interactive Media in Education*, 2005(2):1–24, 2005.

[Ber06] T. Berners-Lee. Linked Data - Design issues, 2006. URL: http://w3.org/DesignIssues/LinkedData, last visited: June 2012.

[Bet03] M.L. Betbeder and P. Tchounikine. SYMBA, a Tailorable Framework to support Collective Activities in a Learning Context. In *Proceedings of the 9th International Workshop on Groupware, CRIWG 2003*, pages 90–98, Grenoble, France, September 2003. Springer-Verlag, LNCS 2806.

[Big82] R. Bigdan and S.K. Biklen. *Qualitative research for education: An introduction to theory and methods.* Allyn and Bacon, Inc., Boston, MA, USA, 1982.

[Biz09] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[Bla09] A. del Blanco, J. Torrente, P. Moreno-Ger, and B. Fernández-Manjón. A General Architecture for the Integration of Educational Videogames in Standards-compliant Virtual Learning Environments. In *Proceedings of the 9th IEEE International Conference on Advanced Learning Technologies, ICALT 2009*, pages 53–55, Riga, Latvia, July 2009. IEEE Computer Society.

[Bla10] A. del Blanco, J. Torrente, E.J. Marchiori, I. Martínez-Ortiz, P. Moreno-Ger, and B. Fernández-Manjón. Easing Assessment of Game-based Learning with <e-Adventure> and LAMS. In *ACM International Workshop on Multimedia Technologies for Distance Learning, MTDL 2010*, pages 25–30, Florence, Italy, October 2010. ACM.

[Bol07] L. Bollen, A. Harrer, H.U. Hoppe, and W. van Joolingen. A broker Architecture for Integration of Heterogeneous Applications for Inquiry Learning. In *Proceedings of*

the 7th IEEE International Conference on Advanced Learning Technologies, ICALT 2007, pages 15–17, Niigata, Japan, July 2007. IEEE Computer Society.

[Boo99] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, MA, USA, 1999.

[Boo09] A.G. Booth and B.P. Clark. A service-oriented Virtual Learning Environment. *On the Horizon*, 17(3):232–244, 2009.

[Bot05] M.L. Bote-Lorenzo. *Gridcole: A Tailorable Grid Service based System that supports Scripted Collaborative Learning*. PhD Thesis, School of Telecommunication Engineering, Universidad de Valladolid, Spain, September 2005.

[Bot08] M.L. Bote-Lorenzo, E. Gómez-Sánchez, G. Vega-Gorgojo, Y.A. Dimitriadis, J.I. Asensio-Pérez, and I.M. Jorrín-Abellán. Gridcole: A Tailorable Grid Service based System that supports Scripted Collaborative Learning. *Computers and Education*, 51(1):155–172, 2008.

[Bot10] M.L. Bote-Lorenzo, J.I. Asensio-Pérez, E. Gómez-Sánchez, G. Vega-Gorgojo, and C. Alario-Hoyos. A Grid Service-Based Distributed Network Simulation Environment for Computer Networks Education. *Computer Applications in Engineering Education*, pages 1–12, 2010. http://dx.doi.org/10.1002/cae.20435, last visited: June 2012.

[Bou01] G. Bourguin and A. Derycke. Integrating the CSCL Activities into Virtual Campuses: Foundations of a new Infrastructure for Distributed Collective Activities. In *Proceedings of the 1st European Conference on Computer Supported Collaborative Learning, EuroCSCL 2001*, pages 123–130, March 2001.

[Bou06] M. Bourimi. Collaborative Design and Tailoring of Web Based Learning Environments in CURE. In *Proceedings of the 12th International Workshop on Groupware, CRIWG 2006*, pages 421–436, Medina del Campo, Spain, September 2006. Springer-Verlag, LNCS 4154.

[Bow11] M. Bower and M. Wittmann. A Comparison of LAMS and Moodle as Learning Design Technologies - Teacher Education Students' Perspective. *Teaching English with Technology, Special Issue on LAMS and Learning Design*, 11(1):62–80, 2011.

[Bri04] C. Britton and P. Bye. *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems (2nd edition)*. Pearson Addison Wesley, London, UK, 2004.

[Bri06] S. Britain and O. Liber. A Framework for Pedagogical Evaluation of Virtual Learning Environment. Technical report, University of Wales, Bangor, UK, 2006. URL: http://www.leeds.ac.uk/educol/documents/00001237.htm, last visited: June 2012.

[Bud08] J. Buder and D. Bodemer. Supporting controversial CSCL discussions with augmented group awareness tools. *International Journal of Computer-Supported Collaborative Learning*, 3(2):122–139, 2008.

[Cae06a] M. Caeiro-Rodríguez, M. Llamas-Nistal, and L. Anido-Rifón. The PoEML Proposal to Model Services in Educational Modeling Languages. In *Proceedings of the 12th International Workshop on Groupware, CRIWG 2006*, pages 187–202, Medina del Campo, Spain, September 2006. Springer-Verlag, LNCS 4154.

[Cae06b] M. Caeiro, M.J. Marcelino, M. Llamas, L. Anido, and A.J. Mendes. PoEML: A Flexibility Approach for Models of Educational Practices. In *Proceedings of the 8th International Symposium on Computers in Education*, pages 24–26, León, Spain, October 2006.

[Cal03] R.A. Calvo, E. Ghiglione, and R.A. Ellis. The OpenACS E-Learning Infrastructure. In *Proceedings of the 9th Australian World Wide Web Conference, AUSWEB03*, pages 175–183, Gold Coast, Australia, July 2003.

[Cay09] E. Cayirci, C. Rong, W. Huiskamp, and C. Verkoelen. Snow Leopard Cloud: A Multinational Education Training and Experimentation Cloud and Its Security Challenges. In *Proceedings of the 1st International Conference on Cloud Computing, CloudCom 2009*, pages 57–68, Beijing, China, December 2009. Springer Verlag, LNCS 5931.

[Cha07] M.A. Chatti, M. Jarke, and D. Frosch-Wilke. The future of e-learning: a shift to knowledge networking and social software. *International Journal of Knowledge and Learning*, 3(4-5):404–420, 2007.

[Chi02] M-L. Chiu. An organizational view of design communication in design collaboration. *Design Studies*, 23(2):187–210, 2002.

[Cho07] H. Cho, G. Gay, B. Davidson, and A. Ingraffea. Social networks, communication styles, and learning performance in a CSCL community. *Computers & Education*, 49(2):309–329, 2007.

[Col02] G. Collier and R. Robson. What is the Open Knowledge Initiative? Technical report, Eduworks Corporation for O.K.I., USA, 2002. URL: http://web.mit.edu/oki/learn/whtpapers/OKI_white_paper_120902.pdf, last visited: June 2012.

[Col07] J. Cole and H. Foster. *Using Moodle: Teaching with the Popular Open Source Course Management System*. O'Reilly Media, Inc., Sebastopol, CA, USA, 2007.

[Con09] M.A. Conde-González, F.J. García-Peñalvo, M.J. Casany-Guerrero, and M. Alier-Forment. Adapting LMS architecture to the SOA: an Architectural Approach. In *Proceedings of the 4th International Conference on Internet and Web Applications and Services, ICIW 2009*, pages 322–327, Venice/Mestre, Italy, May 2009. IEEE Computer Society.

[Con10] M.A. Conde-González, F.J. García-Peñalvo, M.J. Casany-Guerrero, and M. Alier-Forment. Open Integrated Personal Learning Environment: Towards a New Conception of the ICT-Based Learning Processes. In *Proceedings of the 3rd World Summit on the Knowledge Society, WSKS 2010*, pages 115–124, Corfu, Greece, September 2010. Springer-Verlag.

[Cor05] S. Corich. Is it time to Moodle? In *Proceedings of the 18th Annual Conference of the National Advisory Committee on Computing Qualifications, NACCQ 2005*, pages 155–158, Tauranga, New Zealand, July 2005.

[Cur02] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2):86–93, 2002.

[Dag07] D. Dagger, A. O'Connor, S. Lawless, E. Walsh, and V.P. Wade. Service-Oriented E-Learning Platforms: From monolithic Systems to Flexible Services. *IEEE Internet Computing*, 11(3):28–35, 2007.

[Dal03] J. Dalziel. Implementing Learning Design: The Learning Activity Management System (LAMS). In *Proceedings of the Australasian Society for Computers in Learning in Tertiary Education, ASCILITE 2003*, pages 593–596, Adelaide, Australia, December 2003.

[Dal05] J. Dalziel. LAMS Overview. Technical report, MELCOE, Australia, 2005. URL: http://lamsinternational.com/CD/html/resources/summaries/LAMS_Overview.doc, last visited: June 2012.

[Dal06] J. Dalziel. Lesson from LAMS for IMS Learning Design. In *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies, ICALT 2006*, pages 1101–1102, Kerkrade, The Netherlands, July 2006. IEEE Computer Society.

[Dal07] J. Dalziel. Imagining and developing a system for reusable learning designs: lessons from LAMS. *International Journal of Continuing Engineering Education and Life Long Learning*, 17(1):33–42, 2007.

[Dal10] J. Dalziel. Visualising Learning Design in LAMS: A Historical View. *Teaching English with Technology, Special Issue on LAMS and Learning Design*, 11(1):19–34, 2010.

[Dew01] P. Dewan. An integrated approach to designing and evaluating collaborative applications and infrastructures. *Computer Supported Cooperative Work*, 10(1):75–111, 2001.

[Dha95] H. Dhama. Quantitative models of cohesion and coupling in software. *Journal of Systems and Software*, 29(1):65–74, 1995.

[Dil99] P. Dillenbourg. *Collaborative Learning: cognitive and computational approaches.* Elsevier Science, Oxford, UK, 1999.

[Dil00] P. Dillenbourg. Virtual Learning Environments. In *Proceedings of the EUN Conference. Learning in The New Millennium: Building New Education Strategies for Schools*, pages 1–30, Brussels, Belgium, March 2000.

[Dil02a] P. Dillenbourg. *Over-scripting CSCL: The risks of blending collaborative learning with instructional design.* In P.A. Kirschner (ed.), Three worlds of CSCL. Can we support CSCL?. Open University of The Netherlands, Heerlen, The Netherlands, pages 61-91, 2002.

[Dil02b] P Dillenbourg, D.K. Schneider, and P. Synteta. Virtual Learning Environments. In *Proceedings of the 3rd Hellenic Conference Information & Communication Technologies in Education*, pages 3–18, Rhodes, Greece, September 2002.

[Dil07] P. Dillenbourg and P. Tchounikine. Flexibility in macro-scripts for CSCL. *Journal of Computer Assisted Learning*, 23(1):1–13, 2007.

[Dil09] P. Dillenbourg, S. Järvelä, and F. Fischer. *Technology-Enhanced Learning*, chapter 1. The Evolution of Research on Computer-Supported Collaborative Learning: From Design to Orchestration, pages 3–19. Springer Verlag, 2009.

[Dod08] J.M. Dodero and E. Ghiglione. ReST-Based Web Access to Learning Design Services. *IEEE Transactions on Learning Technologies*, 1(3):190–195, 2008.

[Dou99] M. Dougiamas. Developing tools to foster online educational dialogue. In *Proceedings of the 8th Annual Teaching Learning Forum*, pages 119–123, Perth, Australia, February 1999. URL: http://otl.curtin.edu.au/tlf/tlf1999/dougiamas.html, last visited: June 2012.

[Dou00] M. Dougiamas. Improving the effectiveness of tools for Internet based education. In *Proceedings of the 9th Annual Teaching Learning Forum*, Perth, Australia, February 2000. URL: http://lsn.curtin.edu.au/tlf/tlf2000/dougiamas.html, last visited: June 2012.

[Dou03] M. Dougiamas and P.C. Taylor. Moodle: Using learning communities to create an open source course management system. In *Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications, ED-MEDIA 2003*, pages 171–178, Honolulu, HI, USA, June 2003. AACE.

[Dun03] S. Dunn. Return to SENDA? Implementing Accessibility for Disabled Students in Virtual Learning Environments in UK further and higher education. Technical report, City University, London, UK, 2003. URL: http://saradunn.net/VLEreport, last visited: June 2012.

[Dun09] M. Dunleavy, C. Dede, and R. Mitchell. Affordances and Limitations of Immersive Participatory Augmented Reality Simulations for Teaching and Learning. *Journal of Science Education and Technology*, 18(1):7–22, 2009.

[Duq05] P. Duquenoy. *Ethics of computing.* In J. Berleur, and C. Avgerou, (eds.), Perspectives and policies on ICT in society. IFIP Advances in Information and Communication Technology (179). International Federation for Information Processing, Boston, USA, pages 159-170, 2005.

[Dvo11] R. Dvorak. *Moodle for Dummies*, chapter 1. Discovering Moodle and What You Can Do, pages 11–22. Wiley Publishing, Inc., Hoboken, NJ, USA, 2011.

[Eck95] W.W. Eckerson. Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. *Open Information Systems*, 10(3):1–20, 1995.

[Ele11] ElearningForce. Sharepoint LMS Feature List, version 2.1 / 3.0, 2011. URL: http://sharepointlms.com/images/sharepointlms_featurelist_2.1and3.0.pdf, last visited: June 2012.

[Ell91] C.A. Ellis, S.J. Gibbs, and G. Rein. Groupware: Some Issues and Experiences. *Communications of the ACM*, 34(1):39–58, 1991.

[Era04] M. Eraut. Informal Learning in the Workplace. *Studies in Continuing Education*, 26(2):247–273, 2004.

[Far05] J. Farmer and I. Dolphin. Sakai: eLearning and More. In *Proceedings of the 11th European University Information Systems Congress, EUNIS 2005*, pages 1–5, Manchester, UK, January 2005. Citeseer.

[Fer03] N. Ferguson and B. Schneier. *Practical Cryptography.* John Wiley & Sons, Inc., New York, NY, USA, 2003.

[Fie00] R.T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD Thesis, University of California, Irvine, CA, USA, 2000.

[Fie07] S. Fiedler. Getting beyond centralized technologies in higher education, Part 1. In *Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications, ED-MEDIA 2007*, pages 1340–1346, Chesapeake, VA, USA, June 2007. AACE.

[Fol06] G. Folkestad. Formal and informal learning situations or practices vs formal and informal ways of learning. *British Journal of Music Education*, 23(2):135–145, 2006.

[Fon09] J. Fontenla-González, M. Caeiro-Rodríguez, and M. Llamas-Nistal. Towards a Generalized Architecture for the Integration of Tools in LMSs. *International Journal of Emerging Technologies in Learning (iJET)*, 4(S1):6–11, 2009.

[Fos98] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1998.

[Fue08] L. de la Fuente-Valentín, Y. Miao, A. Pardo, and C. Delgado-Kloos. A Supporting Architecture for Generic Service Integration in IMS Learning Design. In *Proceedings of the 3rd European conference on Technology Enhanced Learning, ECTEL 2008*, pages 467–473, Maastricht, The Netherlands, September 2008. Springer-Verlag, LNCS 5192.

[Fue11] L. de la Fuente-Valentín, A. Pardo, and C. Delgado-Kloos. Generic Service Integration in Adaptive Learning Experiences using IMS Learning Design. *Computers & Education*, 57(1):1160–1170, 2011.

[Gam95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, Reading, MA, USA, 1995.

[Gar05] J.J. Garrett. Ajax: A New Approach to Web Applications. Technical report, Adaptive Path, San Francisco, CA, USA, 2005. URL: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications, last visited: May 2012.

[Geo95] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[Ghi06] E. Ghiglione and J. Dalziel. Design principles for LAMS version 2 and the LAMS "Tools Contract". In *Proceedings of the TenCompetence Conference Workshop*, pages 1–15, Barcelona, Spain, June 2006.

[Gif99] B.R. Gifford and N.D. Enyedy. Activity Centered Design: Towards a Theoretical Framework for CSCL. In *Proceedings of the 3rd International Conference on Computer-Supported Collaborative Learning, CSCL 1999*, pages 189–196, Palo Alto, CA, USA, December 1999. ISLS.

[Gla95] R.L. Glass. A Structure-Based Critique of Contemporary Computing Research. *Journal of Systems and Software*, 28(1):3–7, 1995.

[Gom09] E. Gómez-Sánchez, M.L. Bote-Lorenzo, I.M. Jorrín-Abellán, G. Vega-Gorgojo, J.I. Asensio-Pérez, and Y. Dimitriadis. Conceptual framework for design, technological support and evaluation of collaborative learning. *International Journal of Engineering Education*, 25(3):557–568, 2009.

[Gre09] L. Grewe. *OpenSocial Network Programming.* Wrox Press Ltd., Birmingham, UK, 2009.

[Gru92] J. Grundin. Utility and usability: research issues and development context. *Interacting with Computers*, 4(2):209–217, 1992.

[Gul08] S. Gulati. Technology-enhanced learning in developing nations: A review. *International Review of Research in Open and Distance Learning*, 9(1):1–16, 2008.

[Gut09] E. Gutiérrez, M.A. Trenas, J. Ramos, F. Corbera, and S. Romero. A new Moodle module supporting automatic verification of VHDL-based assignments. *Computers & Education*, 54(2):562–577, 2009.

[Hak08] K. Hakkarainen. Three generations of technology-enhanced learning. *British Journal of Educational Technology*, 40(5):879–888, 2008.

[Har06] M. van Harmelen. Personal Learning Environments. In *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies, ICALT 2006*, pages 815–816, Kerkrade, The Netherlands, July 2006. IEEE Computer Society.

[Har07a] D. Hardt, J. Bufu, and J. Hoyt. OpenID Attribute Exchange 1.0 - Final, 2007. URL: http://openid.net/specs/openid-attribute-exchange-1_0.html, last visited: June 2012.

[Har07b] A. Harrer, L. Kobbe, and N. Malzahn. Conceptual and Computational Issues in the Formalization of Collaboration Scripts. In *Proceedings of the 8th International Conference on Computer-Supported Collaborative Learning, CSCL 2007*, pages 280–282, Rutgers, NJ, USA, July 2007. ISLS.

[Has09] M. Häsel and K. Rieke. OpenSocial. *Informatik Spektrum*, 32(3):255–259, 2009.

[Has11] M. Häsel. OpenSocial: An Enabler for Social Applications on the Web. *Communications of the ACM*, 54(1):139–144, 2011.

[Her06a] D. Hernández-Leo, E.D. Villasclaras-Fernández, J.I. Asensio-Pérez, Y. Dimitriadis, and S. Retalis. CSCL Scripting Patterns: Hierarchical Relationships and Applicability. In *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies, ICALT 2006*, pages 388–392, Kerkrade, The Netherlands, July 2006. IEEE Computer Society.

[Her06b] D. Hernández Leo, E.D. Villasclaras Fernández, I.M. Jorrín Abellán, J.I. Asensio Pérez, Y. Dimitriadis, I. Ruiz Requies, and B. Rubia Avi. Collage, a collaborative learning design editor based on patterns. *Educational Technology & Society, Special Issue on Learning Design*, 9(1):58–71, 2006.

[Her07a] D. Hernández-Leo, M.L. Bote-Lorenzo, J.I. Asensio-Pérez, E. Gómez-Sánchez, E.D. Villasclaras-Fernández, I.M. Jorrín-Abellán, and Y.A. Dimitriadis. Free- and Open-Source Software for a Course on Network Management: Authoring and Enactment of Scripts Based on Collaborative Learning Strategies. *IEEE Transactions on Education*, 50(4):292 –301, 2007.

[Her07b] D. Hernández-Leo. *A pattern-based design process for the creation of CSCL macroscripts computationally represented with IMS LD.* PhD Thesis, School of Telecommunications Engineering, Universidad de Valladolid, Spain, April 2007.

[Her11] D. Hernández-Leo, R. Nieves, E. Arroyo, A. Rosales, J. Melero, P. Moreno, and J. Blat. Orchestration Signals in the Classroom: Managing the Jigsaw Collaborative Learning Flow. In *Proceedings of 6th European Conference on Technology Enhanced Learning, ECTEL 2011*, pages 153–165, Palermo, Italy, September 2011. Springer-Verlag, LNCS 6964.

[How03] T.A. Howes, M.C. Smith, and G.S. Good. *Understanding and Deploying LDAP Directory Services.* Addison-Wesley Longman Publishing Co., Boston, MA, USA, 2003.

[Iba11] M.B. Ibañez, C. Delgado-Kloos, D. Leony, J.J. García-Rueda, and D. Maroto. Learning a Foreign Language in a Mixed-Reality Environment. *IEEE Internet Computing*, 15(6):44–47, 2011.

[IET10] IETF, Internet Engineering Task Force. The OAuth 1.0 Protocol, 2010. URL: http://tools.ietf.org/html/rfc5849, last visited: June 2012.

[IMS03] IMS Global Learning Consortium. IMS Learning Design specification, 2003. URL: http://imsglobal.org/learningdesign, last visited: June 2012.

[IMS06a] IMS Global Learning Consortium. IMS Meta-data Best Practice Guide, 2006. URL: http://imsglobal.org/metadata, last visited: June 2012.

[IMS06b] IMS Global Learning Consortium. IMS Question & Test Interoperability Specification. Version 2.1 Public Draft (revision 2) Specification, 2006. URL: http://imsglobal.org/question, last visited: June 2012.

[IMS06c] IMS Global Learning Consortium. IMS Tool Interoperability Guidelines. Versión 1.0, 2006. URL: http://imsglobal.org/ti, last visited: June 2012.

[IMS07] IMS Global Learning Consortium. IMS Content Packaging Specification Primer. Version 1.2 Public Draft v2.0, 2007. URL: http://imsglobal.org/content/packaging, last visited: June 2012.

[IMS10a] IMS Global Learning Consortium. IMS GLC Learning Information Services Specification Primer Version 2.0, 2010. URL: http://imsglobal.org/lis, last visited: June 2012.

[IMS10b] IMS, IMS Global Learning Consortium. IMS GLC Learning Tools Interoperability Basic LTI Implementation Guide Version 1.0.1 Public Draft, 2010. URL: http://imsglobal.org/lti, last visited: June 2012.

[IMS11] IMS Global Learning Consortium. IMS GLC Common Cartridge Profile. Version 1.1 Final Specification, 2011. URL: http://imsglobal.org/cc, last visited: June 2012.

[IMS12] IMS Global Learning Consortium. IMS GLC Learning Tools Interoperability Implementation Guide. Final Version 1.1, 2012. URL: http://imsglobal.org/lti, last visited: June 2012.

[Iso10] S. Isotani, R. Mizoguchi, A. Inaba, and M Ikeda. The foundations of a theory-aware authoring tool for CSCL design. *Computers & Education*, 54(4):809–834, 2010.

[Jac99] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Pearson Education Inc., Boston, MA, USA, 1999.

[Jav11] Java Community Process. Java Network Launching Protocol and API, 2011. URL: http://jcp.org/aboutJava/communityprocess/maintenance/jsr056/index6.html, last visited: June 2012.

[Joh04] P. John and R. Sutherland. Teaching and learning with ICT: new technology, new pedagogy? *Education and Communication and Information*, 4(1):101–109, 2004.

[Jon99] D.H. Jonassen, W.S. Pfeiffer, and B.G. Wilson. *Learning with technology: A constructivist perspective*. Prentice Hall, Upper Saddle River, NJ, USA, 1999.

[Jon05] C. Jones, L. Dirckinck-Holmfeld, and B. Lindström. A relational, indirect, meso-level approach to CSCL design in the next decade. *International Journal of Computer-Supported Collaborative Learning*, 1(1):35–56, 2005.

[Jor07] I.M. Jorrín-Abellán and B. Rubia-Avi. What the eye doesn't see: An Inquiry (cowiki) based learning case study. In *Proceedings of the 3rd International Congress of Qualitative Inquiry*, page 266, Urbana-Champaign, IL, USA, May 2007.

[Jor09] I.M. Jorrín-Abellán, R.E. Stake, and A. Martínez-Monés. The Needlework in evaluating a CSCL system: The Evaluand oriented Responsive Evaluation Model. In *Proceedings of the 9th International conference on Computer-Supported Collaborative Learning, CSCL 2009*, pages 68–72, Rhodes, Greece, June 2009. ISLS.

[Kat10] D. Katsifli. The impact of Blackboard software on education globally over the past 10 years, with a focus on the measurable benefits from using Blackboard Learn software and related technologies. Technical report, Blackboard Inc., USA, 2010. URL: http://lms.unimelb.edu.au/elo/resources/The_impact_of_Blackboard_software-_on_education_globally_(20100204W).pdf, last visited: June 2012.

[Kit96a] B. Kitchenham. DESMET: A method for evaluating Software Engineering methods and tools. Technical report, Department of Computer Science, University of Keele, UK, 1996. URL: http://www.osel.co.uk/desmet.pdf, last visited: June 2012.

[Kit96b] B. Kitchenham. Evaluating software engineering methods and tool part 1: The evaluation context and evaluation methods. *ACM SIGSOFT Software Engineering Notes*, 21(1):11–14, 1996.

[Kit97a] B. Kitchenham. Evaluating Software Engineering Methods and Tool Part 7: Planning Feature Analysis Evaluation. *ACM SIGSOFT Software Engineering Notes*, 22(4):21–24, 1997.

[Kit97b] B. Kitchenham, S. Linkman, and D. Law. DESMET: a methodology for evaluating software engineering methods and tools. *Computing & Control Engineering Journal*, 8(3):120–126, 1997.

[Kni06] H. Kniberg. Scrum and XP from the Trenches. How we do Scrum. Technical report, Crisp, Sweden, 2006. URL: http://www.metaprog.com/csm/ScrumAndXpFromTheTrenches.pdf, last visited: June 2012.

[Koe05] C. Koeber. Introducing Multimedia Presentations and a Course Website to an Introductory Sociology Course: How Technology Affects Student Perceptions of Teaching Effectiveness. *Teaching Sociology*, 33(3):285–300, 2005.

[Kop04] R. Koper and B. Olivier. Representing the Learning Design of Units of Learning. *Educational Technology & Society*, 7(3):97–111, 2004.

[Kop05] R. Koper and C. Tattersall (eds.). *Learning Design, a Handbook on Modelling and Delivering Networked Education and Training*. Springer, Heidelberg, Germany, 2005.

[Kos96] T. Koschmann. *CSCL: theory and practice of an emerging paradigm*. Lawrence Erlbaum, Malwah, NJ, USA, 1996.

[Kra88] G.E. Krasner and S.T. Pope. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *Journal of Object Oriented Program*, 1(3):26–49, 1988.

[Kru04] P. Kruchten. *The Rational Unified Process: An Introduction (3rd edition)*. Pearson Education Inc., Boston, MA, USA, 2004.

[Kus04] S. Kusumoto, F. Matukawa, K. Inoue, S. Hanabusa, and Y. Maegawa. Estimating effort by use case points: method, tool and case study. In *Proceedings of the 10th IEEE International Symposium on Sofware Metrics, METRICS 2004*, pages 292–299, Chicago, IL, USA, September 2004. IEEE Computer Society.

[Lar02] C. Larman. *Applying UML and patterns: An introduction to Object-Oriented Analysis and Design and the Unified Process (2nd edition)*. Prentice Hall Professional, Upper Saddle River, NJ, USA, 2002.

[Leb09] M. Lebrun, F. Docq, and D. Smidts. Claroline, an Internet Teaching and Learning Platform to Foster Teachers' Professional Development and Improve Teaching Quality: First Approaches. *Association for the Advancement of Computing in Education (AACE) Journal*, 17(4):347–362, 2009.

[Len07] A. Lenhart and M. Madden. Social Networking Websites and Teens: An Overview. Technical report, PEW Internet and American Life Project, USA, 2007. URL: http://pewinternet.org/PPF/r/198/report_display.asp, last visited: June 2012.

[Leu01] B. Leuf and W. Cunningham. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley, London, UK, 2001.

[Lik32] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55, 1932.

[Lip02] L. Lipponen. Exploring Foundations for Computer-Supported Collaborative Learning. In *Proceedings of the 5th Computer-Supported Collaborative Learning 2002 Conference, CSCL 2002*, pages 72–81, Boulder, CO, USA, January 2002. Lawrence Erlbaum Associates.

[Liu10] C. Liu and X. Zhang. Building E-Learning Platform in Vocational and Higher Training College with Claroline. In *Proceedings of the 2010 International Conference on E-Business and E-Government, ICEE 2010*, pages 3779–3782, Guangzhou, China, May 2010. IEEE Computer Society.

[Liv08] D. Livingstone and J. Kemp. Integrating Web-Based and 3D Learning Environments: Second Life Meets Moodle. *UPGRADE: The European Journal for the Informatics Professional*, 9(3):8–14, 2008.

[Loc08] L. Lockyer and J. Patterson. Integrating Social Networking Technologies in Education: A Case Study of a Formal Learning Environment. In *Proceedings of the 8th IEEE International Conference on Advanced Learning Technologies, ICALT 2008*, pages 529–533, Santander, Spain, July 2008. IEEE Computer Society.

[Lou12] J. Louvel, T. Templier, and T. Boileau. *Restlet in Action*. Manning Publications Co., Greenwhich, CT, USA, 2012.

[Mad09] C. Madgea, J. Meekb, J. Wellensc, and T. Hooleyd. Facebook, social integration and informal learning at university: "It is more for socialising and talking to friends about work than for actually doing work". *Learning, Media and Technology, Special Issue on Learning and social software, researching the realities*, 34(2):141–155, 2009.

[Mar83] J. Martin. *Managing the data-base environment*. Prentice-Hall, Upper Saddle River, NJ, USA, 1983.

[Mar03] A. Martínez-Monés, Y Dimiatriadis, B. Rubia-Aví, E. Gómez-Sánchez, and P. de la Fuente. Combining qualitative evaluation and social network analysis for the study of classroom social interactions. *Computers & Education*, 41(3):353–368, 2003.

[Mar04] C. Martel, C. Ferraris, B. Caron, T. Carron, G. Chabert, C. Courtin, L. Gagnière, J.C. Marty, and L. Vignollet. A Model for CSCL Allowing Tailorability: Implementation in the Electronic Schoolbag Groupware. In *Proceedings of the X International Workshop on Groupware, CRIWG 2004*, pages 322–338, San Carlos, Costa Rica, September 2004. Springer, LNCS 3198.

[Mar08a] A. Martínez-Monés, S. Villagrá-Sobrino, R. Santos-Fernández, R. Anguita-Martínez, and I.M. Jorrín-Abellán. Social network analysis support for an IBL wiki-based course. In *Proceedings of the Workshop on Real-Time methods at International Conference of the Learning Sciences, ICLS 2008*, pages 1–3, Utrecht, The Netherlands, June 2008.

[Mar08b] M. Martínez and S. Jagannathan. Moodle: A Low-Cost Solution for Successful e-Learning. *Learning Solutions Magazine [Online]*, 2008. URL:

http://www.learningsolutionsmag.com/articles/71/moodle-a-low-cost-solution-for-successful-e-learning, last visited: June 2012.

[Mat97] Y. Matsubara, S. Toihara, Y. Tsukinari, and M. Nagamachi. Virtual Learning Environment for Discovery Learning and Its Application on Operator Training. *IEICE Transactions on Information and Systems*, E80-D(2):176–188, 1997.

[Maz05] D. Mazurek. CAS Protocol Version 1.0, 2005. URL: http://jasig.org/cas/protocol, last visited: June 2012.

[Mia05] Y. Miao, K. Hoeksema, H.U. Hoppe, and A Harrer. CSCL scripts: Modelling features and potential use. In *Proceedings of the Computer-Supported Collaborative Learning Conference 2005: CSCL 2005*, pages 423–432, Taipei, Taiwan, May 2005. Lawrence Erlbaum Associates.

[Mil11] D.E. Millard, H.C. Davis, Y. Howard, P. McSweeney, C. Yorke, H. Solheim, and D. Morris. Towards an Institutional PLE. In *Proceedings of the 2nd International Conference of Personal Learning Environments, PLE 2011*, pages 1–14, Southampton, UK, July 2011.

[Mon12] R. Mondéjar-Andreu, P. García-López, E. Fernández-Casado, and C. Pairot-Gavaldà. TaKo: Providing transparent collaboration on single-user applications. *Computer Languages, Systems & Structures*, 38(1):108–121, 2012.

[Mor95] A. Morch. Three levels of end-user tailoring: customization, integration and extension. In *Proceedings of the 3rd Decennial Aarhus Conference*, pages 41–45, Aarhus, Denmark, August 1995.

[Mor98] D.L. Morgan. *The Focus Group Guidebook*. SAGE, Thousand Oaks, CA, USA, 1998.

[Mor00] A. Morch and N.D. Mehandjiev. Tailoring as collaboration: the mediating role of multiple representations and application units. In *Proceedings of the ACM 2000 Conference on Computer Supported Cooperative Work, CSCW 2000*, pages 75–100, Philadelphia, PA, USA, December 2000. ACM.

[Mor03] H. Morris and A. Rippin. Virtual Learning Environments in Business and Management: A Review of Some Recent Developments. *International Journal of Management Education*, 3(2):23–30, 2003.

[Mor04] R.L. Morgan, S. Cantor, S. Carmody, W. Hoehn, and K. Klingenstein. Federated Security: The Shibboleth Approach. *EDUCAUSE Quarterly*, 27(4):12–17, 2004.

[Mue11] D. Mueller and S. Strohmeier. Design Characteristics of Virtual Learning Environments: State of Research. *Computers & Education*, 57(4):2505–2516, 2011.

[Mun12a] J.A. Muñoz-Cristóbal, J.I. Asensio-Pérez, L.P. Prieto, I.M. Jorrín-Abellán, Y. Dimitriadis, and A. Martínez-Monés. Helping educators to deploy CSCL scripts into mainstream VLEs that integrate third-party Web and Augmented Reality Tools. In *Proceedings of the Workshop on Digital Ecosystems for Collaborative Learning 2012 held in conjunction with the International Conference of the Learning Sciences, ICLS 2012 (accepted)*, Sydney, Australia, June 2012.

[Mun12b] J.A. Muñoz-Cristóbal, L.P. Prieto, J.I. Asensio-Pérez, I.M. Jorrín-Abellán, and Y. Dimitriadis. Lost in Translation from Abstract Learning Design to ICT Implementation: A Study Using Moodle for CSCL. In *Proceedings of the 7th European Conference on Technology Enhanced Learning, ECTEL 2012 (submitted)*, 2012.

[Nav11] T. Navarrete, P. Santos, D. Hernández-Leo, and J. Blat. QTIMaps: A Model to Enable Web Maps in Assessment. *Educational Technology & Society Journal*, 14(3):203–217, 2011.

[NWG99] NWG, Network Working Group. Hypertext Transfer Protocol – HTTP/1.1, 1999. URL: http://ietf.org/rfc/rfc2616, last visited: June 2012.

[NWG05a] NWG, Network Working Group. The Atom Syndication Format, 2005. URL: http://ietf.org/rfc/rfc4287, last visited: June 2012.

[NWG05b] NWG, Network Working Group. Uniform Resource Identifier (URI): Generic Syntax, 2005. URL: http://tools.ietf.org/html/rfc3986, last visited: June 2012.

[NWG06] NWG, Network Working Group. Scripting Media Types, 2006. URL: http://tools.ietf.org/html/rfc4329, last visited: June 2012.

[NWG11] NWG, Network Working Group. The OAuth 2.0 Authorization Protocol. Internet Draft, 2011. URL: http://tools.ietf.org/html/draft-ietf-oauth-v2-25, last visited: June 2012.

[OAS06] OASIS. Web Service Resource Framework (WSRF) - Primer v1.2. Committee Draft 02, 2006. URL: http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf, last visited: June 2012.

[OLe02] R. O'Leary. Virtual Learning Environments. Technical report, LTSN Generic Centre, University of Bristol, UK, 2002. URL: ftp://www.bioscience.heacademy.ac.uk/Resources/gc/elearn2.pdf, last visited: June 2012.

[ORe07] T. O'Reilly. What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. Technical report, O'Reilly Media Inc., Sebastopol, CA, USA,

2007. URL: http://oreilly.com/web2/archive/what-is-web-20.html, last visited: June 2012.

[Ort90] J.D. Orton and K.E. Weick. Loosely Coupled Systems: A Reconceptualization. *Academy of Management Review*, 15(2):203–223, 1990.

[Osg03] R.T. Osguthorpe and C.R. Graham. Blended Learning Environments: Definitions and Directions. *Quarterly Review of Distance Education*, 4(3):227–233, 2003.

[Osu99] C.A. Osuna-Gómez. *DELFOS: An educational telematic framework based on levels oriented to cooperative learning situations*. PhD Thesis, School of Telecommunication Engineering, Universidad de Valladolid, Spain, December 1999.

[Ous89] J.K. Ousterhout. *Tcl: An Embeddable Command Language*. University of California, Computer Science Division, Berkeley, CA, USA, 1989.

[Pal07] L. Palomino-Ramírez, A. Martínez-Monés, M.L. Bote-Lorenzo, J.I. Asensio-Pérez, and Y.A. Dimitriadis. Data Flow between Tools: Towards a Composition-Based Solution for Learning Design. In *Proceedings of the 7th IEEE International Conference on Advanced Learning Technologies, ICALT 2007*, pages 354–358, Niigata, Japan, July 2007. IEEE Computer Society.

[Pal08] L. Palomino-Ramírez, M.L. Bote-Lorenzo, J.I. Asensio-Pérez, and Y. Dimitriadis. LeadFlow4LD: Learning and Data Flow Composition-based Solution for Learning Design in CSCL. In *Proceedings of the 14th International Workshop on Groupware, CRIWG 2008*, pages 266–280, Omaha, NE, USA, September 2008. Springer-Verlag, LNCS 5411.

[Pap03] M.P. Papazoglou and D. Georgakopoulos. Service-Oriented Computing. *Communications of the ACM*, 10(46):24–28, 2003.

[Pat05] H. Patzold. Increasing value without increasing effort? The use of WebCT in accompanying face-to-face lectures under the constraint of low budget. *Journal of Distance Education*, 20(2):78–84, 2005.

[Pau08] C. Pautasso, O. Zimmermann, and F. Leymann. RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. In *Proceeding of the 17th international conference on World Wide Web, WWW 2008*, pages 805–814, Beijing, China, April 2008. ACM.

[Pau09] C. Pautasso and E. Wilde. Why is the Web Loosely Coupled?: A Multi-Faceted Metric for Service Design. In *Proceedings of the 18th international conference on World Wide Web, WWW 2009*, pages 911–920, Madrid, Spain, April 2009. ACM.

[Pel01] W.J. Pelgrum. Obstacles to the Integration of ICT in Education: Results from a Worldwide Education Assessment. *Computers & Education*, 37(2):163–178, 2001.

[Pen07] P.C. Pendharkar and J.A. Rodger. An empirical study of the impact of team size on software development effort. *Information Technology and Management*, 8(4):253–262, 2007.

[Per10] R. Pérez-Rodríguez, M. Caeiro-Rodríguez, J. Fontela-González, and L.A. Anido-Rifón. Orchestrating Groupware in Engineering Education. In *Proceedings of the 40th IEEE Annual Frontiers in Education Conference, FIE 2010*, pages F3D1–F3D6, Washington DC, USA, October 2010. IEEE Computer Society.

[Per11] M. Pérez-Sanagustín. *Operationalization of collaborative blended learning scripts: a model, computational mechanisms and experiments*. PhD Thesis, Universitat Pompeu Fabra, Barcelona, Spain, July 2011.

[Pit03] M. Pittinsky. Blackboard Building Blocks 2003 Overview White Paper. Technical report, Blackboard Inc., USA, 2003. URL: http://library.blackboard.com/docs/developer/White_Paper_2003.pdf, last visited: June 2012.

[Pos00] S. Poslad, P. Buckle, and R. Hadingham. Open Source, Standards and Scaleable Agencies. In *Proceedings of Autonomous Agents 2000 Workshop on Infrastructure for Scalable Multi-agent Systems*, pages 296–303, Barcelona, Spain, June 2000. Springer-Verlag, LNCS 1887.

[Pri11] L.P. Prieto, S. Villagrá-Sobrino, I.M Jorrín-Abellán, A. Martínez-Monés, and Y. Dimitriadis. Recurrent routines: Analyzing and supporting orchestration in technology-enhanced primary classrooms. *Computers & Education*, 57(1):1214–1227, 2011.

[Reg09] L.M. Regueras, E. Verdú, J.P. de Castro, M.A. Pérez, and M.J. Verdú. A Proposal of User Interface for a Distributed Asynchronous Remote Evaluation System: An Evolution of the QUESTOURnament Tool. In *Proceedings of the 9th IEEE International Conference of Advanced Learning Technologies, ICALT 2009*, pages 75–77, Riga, Latvia, July 2009. IEEE Computer Society.

[Ric07] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media, Inc., Sebastopol, CA, USA, 2007.

[Ric10] W. Richardson. *Blogs, Wikis, Podcasts, and Other Powerful Web Tools for Classrooms*. Corwin Press, Thousand Oaks, CA, USA, 2010.

[Rip02] B.D. Ripley. Statistical Methods Need Software: A View of Statistical Computing. In *International Conference of the Royal Statistical Society. Opening lecture*, Plymouth, UK, September 2002.

[Ris00] L. Rising and N.S. Janoff. The Scrum Software Development Process for Small Teams. *IEEE Software*, 17(4):26–32, 2000.

[Rob97] M.D. Roblyer, J. Edwards, and M.A. Havriluk. *Integrating educational technology into teaching*. Prentice Hall/Merrill College Publishing Company, Columbus, OH, USA, 1997.

[Rob00] R. Robson. Report on Learning Technology Standards. In *Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications, ED-MEDIA 2000*, pages 936–941, Montreal, Canada, June 2000. AACE.

[Rob11] G. Robles, J.M. González-Barahona, and J. Fernández-González. Implementing gymkhanas with android smartphones: A multimedia m-learning game. In *Proceedings of the IEEE Global Engineering Education Conference, EDUCON 2011*, pages 960–968, Amman, Jordan, April 2011. IEEE Computer Society.

[Rod11] M.J. Rodriguez-Triana, A. Martínez-Monés, and J.I. Asensio-Pérez. Monitoring Collaboration in Flexible and Personal Learning Environments. *Interaction Design and Architecture(s) Journal*, 11-12:51–63, 2011.

[Rod12] M.J. Rodriguez-Triana, A. Martínez-Monés, J.I. Asensio-Pérez, and Y. Dimitriadis. Towards a monitoring-aware learning design process. In *Proceedings of the 18th Conference on Collaboration and Technology, CRIWG 2012 (accepted)*, Duisburg, Germany, September 2012.

[Ros11] M.J. Rosenberg. *E-Learning: Strategies for Delivering Knowledge in the Digital Age*. McGraw Hill, Inc. New York, NY, USA, 2011.

[Rui12a] A. Ruiz-Calleja, G. Vega-Gorgojo, J.I. Asensio-Pérez, M.L. Bote-Lorenzo, E. Gómez-Sánchez, and C. Alario-Hoyos. A linked data approach for the discovery of educational ICT tools in the Web of Data. *Computers & Education (in press)*, 59(3):952-962, 2012.

[Rui12b] A. Ruiz-Calleja, G. Vega-Gorgojo, E. Gómez-Sánchez, J.I. Asensio-Pérez, C. Alario-Hoyos, and M.L. Bote-Lorenzo. Automatic retrieval of educational ICT tool descriptions from the Web of Data. In *Proceedings of the 12th IEEE International Conference on Advanced Learning Technologies, ICALT 2012 (accepted)*, Rome, Italy, July 2012. IEEE Computer Society.

[San07] O.C. Santos, J.G. Boticario, E. Raffene, and R. Pastor. Why using dotLRN? UNED use cases. In *Proceedings of the Free, Libre, Open Source Software International Conference 2007, FLOSS 2007*, pages 195–211, Xerez, Spain, March 2007.

[San11] M.J. Sánchez-Franco, A.F. Villarejo-Ramos, and F.A. Martín-Velicia. Social integration and post-adoption usage of Social Network Sites An analysis of effects on learning performance. *Procedia - Social and Behavioral Sciences*, 15(1):256–262, 2011.

[Say05] R. Sayre. Atom: the standard in syndication. *IEEE Internet Computing*, 9(4):71–78, 2005.

[Sea09] J.M. Seaman. Online Learning as a Strategic Asset. Volume II: The Paradox of Faculty Voices: Views and Experiences with Online Learning. Technical report, Association of Public and Land-Grant Universities, ASUP, USA, 2009. URL: http://www.aplu.org/document.doc?id=1879, last visited: June 2012.

[Sev08] C. Severance, J. Hardin, and A. Whyte. The coming functionality mash-up in Personal Learning Environments. *Interactive Learning Environments*, 16(1):47–62, 2008.

[Sha96a] M. Shaw and P. Clements. Toward boxology: preliminary classification of architectural styles. In *Joint Proceedings of the 2nd International Software Architecture Workshop, and the International Workshop on Multiple Perspectives in Software Development on SIGSOFT '96 workshops*, pages 50–54, San Francisco, CA, USA, October 1996. ACM.

[Sha96b] M. Shaw and D. Garland. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, Upper Saddle River, NJ, USA, 1996.

[Sha00] M. Sharples. The design of personal mobile technologies for lifelong learning. *Computers & Education*, 34(3-4):177–193, 2000.

[Sim07] M. Simonson. Course Management Systems. *The Quarterly Review of Distance Education*, 8(1):7–9, 2007.

[Sol05] A. Soller, A. Martínez, P. Jermann, and M. Muehlenbrock. From Mirroring to Guiding: A Review of the State of the Art Technology for Supporting Collaborative Learning. *International Journal on Artificial Intelligence in Education*, 15(4):261–290, 2005.

[Sol07] G. Solomon and L. Schrum. *Web 2.0: new tools, new schools*. International Society for Technology in Education (ISTE), Washington DC, USA, 2007.

[Sta06] G. Stahl, T. Koschmann, and D. Suthers. *Computer-Supported Collaborative Learning: An Historical Perspective.* In R.K. Sawyer (ed.), Cambridge handbook of the learning sciences, Cambridge University Press, Cambridge, UK, pages 409-426, 2006.

[Sta10] G. Stahl and F. Hesse. The CSCL field matures. *Computer-Supported Collaborative Learning*, 5(1):1–3, 2010.

[Sti00] M.J. Stiles. Effective Learning and the Virtual Learning Environment. In *Proceedings of the 6th European University Information Systems Congress, EUNIS 2000*, pages 171–180, Poznan, Poland, Abril 2000.

[Sti07] M. Stiles. Death of the VLE?: a challenge to a new orthodoxy. *Journal for the Serials Community*, 20(1):31–36, 2007.

[StL01] S. St. Laurent, J. Johnston, and E. Dumbillt. *Programming Web Services with XML-RPC.* O'Really Inc. Sebastopol, CA, USA, 2001.

[Sur09] P.K. Suri and N. Garg. Software Reuse Metrics: Measuring Component Independence and its applicability in Software Reuse. *International Journal of Computer Science and Network Security*, 9(5):237–248, 2009.

[Tho08] M. Thompson. ICT and development studies: Towards development 2.0. *Journal of International Development*, 20(6):821–835, 2008.

[Tur03] T. Turner, D. Budgen, and P. Brereton. Turning Software into a Service. *Computer*, 36(10):38–44, 2003.

[Uzu06] H. Uzunboylu, F. Ozdamli, and Z. Ozcinar. An Evaluation of Open Source Learning Management Systems According to Learners Tools. *Current Developments in Technology Assisted Education*, 1(1):8–12, 2006.

[Veg08] G. Vega-Gorgojo, M.L. Bote-Lorenzo, E. Gómez-Sánchez, J.I. Asensio-Pérez, Y. Dimitriadis, and I.M. Jorrín-Abellán. Ontoolcole: Supporting Educators in the Semantic Search of CSCL Tools. *Journal of Universal Computer Science (JUCS)*, 14(1):27–58, 2008.

[Vel09] J. Vélez-Reyes. *Pelican, a platform for the design and development of collaborative learning scenarios. Support for dynamic aspects.* PhD Thesis, UNED. Spain, 2009.

[Vig78] L.S. Vigotsky. *Mind in Society: The Development of Higher Psychological Processes.* Harvard University Press, Cambridge, MA, USA, 1978.

[Vig06] L. Vignollet, J-P. David, C. Ferraris, C. Martel, and A. Lejeune. Comparing Educational Modeling Languages on a Case Study. In *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies, ICALT 2006*, pages 1149–1151, Kerkrade, The Netherlands, July 2006. IEEE Computer Society.

[Vig08] L. Vignollet, C. Ferraris, C. Martel, and D. Burgos. A Transversal Analysis of Different Learning Design Approaches. *Journal of Interactive Media in Education*, pages 1–11, 2008. URL: http://jime.open.ac.uk/2008/26, last visited: June 2012.

[Vil09] E.D. Villasclaras-Fernández, D. Hernández-Leo, J.I. Asensio-Pérez, and Y. Dimitriadis. Incorporating Assessment in a Pattern-based Design Process for CSCL scripts. *Computers in Human Behavior*, 25(5):1028–1039, 2009.

[Vil10] E.D. Villasclaras-Fernández. *A design process supported by software authoring tools for the integration of assessment within CSCL scripts.* PhD Thesis, School of Telecommunication Engineering, Universidad de Valladolid, Spain, November 2010.

[Vin97] S. Vinoski. CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications*, 35(2):46–55, 1997.

[Vin02] S. Vinoski. Putting the "Web" into Web Services: Web Services interaction models, part 2. *IEEE Internet Computing*, 6(4):90–92, 2002.

[Vin07] S. Vinoski. REST Eye for the SOA Guy. *Internet Computing, IEEE*, 11(1):82–84, January 2007.

[Vog06] H. Vogten, H. Martens, R. Nadolski, C. Tattersall, P. van Rosmalen, and R. Koper. CopperCore Service Integration - Integrating IMS Learning Design and IMS Question and Test Interoperability. In *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies, ICALT 2006*, pages 378–382, Kerkrade, The Netherlands, July 2006. IEEE Computer Society.

[W3C99] W3C, World Wide Web Consortium. HTML 4.01 Specification, W3C Recommendation, 1999. URL: http://w3.org/TR/html4, last visited: June 2012.

[W3C08] W3C, World Wide Web Consortium. Web Content Accessibility Guidelines (WCAG) 2.0, W3C Recommendation, 2008. URL: http://w3.org/TR/WCAG, last visited: June 2012.

[W3C09a] W3C, World Wide Web Consortium. XForms 1.1, W3C Recommendation, 2009. URL: http://w3.org/TR/xforms, last visited: June 2012.

[W3C09b] W3C, World Wide Web Consortium. Web Forms 2.0, 2009. URL: http://w3.org/TR/web-forms-2, last visited: June 2012.

[W3C11] W3C, World Wide Web Consortium. Widget Packaging and XML Configuration. W3C Recommendation, 2011. URL: http://w3.org/TR/widgets, last visited: June 2012.

[W3C12] W3C, World Wide Web Consortium. HTML5: A vocabulary and associated APIs for HTML and XHTML, Editors' draft 2012, 2012. URL: http://dev.w3.org/html5/spec/single-page.html, last visited: June 2012.

[Wal04] M. Walckiers and T. De Praetere. Online Collaborative learning, eight benefits that make it a must. *Distances et savoirs*, 1(2):53–75, 2004. In French.

[Wal10] J. Walden, M. Doyle, R. Lenhof, and J. Murray. Idea: Java vs. PHP: Security Implications of Language Choice for Web Applications. In *Proceedings of the 2nd International Symposium on Engineering Secure Software and Systems, ESSoS 2010*, pages 61–69, Pisa, Italy, February 2010.

[Wel91] J. Wellington. Newspaper science, school science, friends or enemies? *International Journal of Science Education*, 13(4):363–372, 1991.

[Wel06] M. Weller. VLE 2.0 and future directions in learning environments. In *Proceedings of the 1st International LAMS Conference*, pages 99–106, Sydney, Australia, December 2006.

[Wel07a] M. Weller. *Virtual Learning Environments: Using, Choosing and Developing Your VLE*. Routledge, Oxford, UK, 2007.

[Wel07b] M. Weller and J. Dalziel. Bridging the gap between Web 2.0 and Higher Education. In *Proceedings of the 2nd International LAMS conference*, pages 76–82, Sydney, Australia, January 2007.

[Whi11] S. White and H.C. Davis. Making it Rich and Personal: crafting an institutional personal learning environment. *International Journal of Virtual and Personal Learning Environments (In Press)*, 2(4):23–39, 2011.

[Wil06] S. Wilson, O. Liber, P. Beauvoir, C. Milligan, M. Johnson, and P. Sharples. Personal learning environments: Challenging the dominant design of educational systems. In *Proceedings of the 1st Joint International Workshop on Professional Learning, Competence Development and Knowledge Management - LOKMOL and L3NCD*, pages 68–77, Crete, Greece, October 2006. Springer-Verlag.

[Wil08] S. Wilson, P. Sharples, and D. Griffiths. Distributing education services to personal and institutional systems using Widgets. In *Proceedings of the 1st International*

Workshop on Mashup Personal Learning Environments, MUPPLE 2008, pages 25–32, Maastricht, The Netherlands, September 2008.

[Win93] L. Winner. Upon Opening the Black Box and Finding It Empty: Social Constructivism and the Philosophy of Technology. *Science, Technology, & Human Values*, 18(3):362–378, 1993.

[Xu05] D. Xu, H. Wang, and M. Wang. A Conceptual Model of Personalized Virtual Learning Environments. *Expert Systems with Applications*, 29(3):525–534, 2005.

[Zel02] M.V. Zelkowitz and D.R. Wallace. Experimental models for validating technology. *Computer*, 31(5):23–31, 2002.

Summary in Spanish

Resumen en español

Preámbulo

El resto de este documento constituye el resumen en español de la tesis titulada *GLUE!: An Architecture for the Integration of External Tools in Virtual Learning Environments* (GLUE!: Una Arquitectura para la Integración de Herramientas Externas en Entornos de Aprendizaje Virtual). En primer lugar, se presenta en este resumen el índice correspondiente a la versión en inglés traducido al español, con el objetivo de facilitar la consulta y la ampliación de detalles en las diferentes secciones de la tesis en su versión original. Posteriormente, se resume el contenido de cada capítulo, haciendo especial énfasis en las contribuciones que de ellos se derivan. Es importante tener en cuenta, que todas las referencias utilizadas en estos resúmenes se refieren al listado de referencias que aparece en la versión en inglés. Debido a su contenido más técnico, no se incluye ningún resumen de los anexos que se presentan al final de la tesis.

El primer capítulo, que traduce prácticamente en su totalidad la versión en inglés, contextualiza el problema de investigación que se aborda en la tesis, para luego definir los objetivos y las contribuciones esperadas. Además, en este capítulo se describe brevemente la metodología de investigación global seguida, la cual está compuesta por cuatro fases que se recorren de forma iterativa. Finalmente, este capítulo introductorio ofrece al lector una estructura global del documento, resumiendo brevemente lo que puede esperar de cada uno de los capítulos restantes.

El segundo capítulo resume, de manera muy compacta, los requisitos de los principales actores interesados en integrar herramientas externas en Entornos de Aprendizaje Virtual (VLE - *Virtual Learning Environments*). A continuación, se repasan brevemente las alternativas que pueden tomarse sobre diferentes decisiones de diseño, con el objetivo de cumplir dichos requisitos. En la versión en inglés se ofrece una discusión más profunda sobre cómo se llega a estos requisitos, a partir de una análisis pormenorizado del contexto en el que se enmarca la tesis. Además, en la versión original, también se estudian las alternativas a las decisiones de diseño tomadas por otras propuestas de integración.

El tercer capítulo ofrece una visión general de la arquitectura de integración propuesta, y un breve resumen de los contratos definidos para esta arquitectura. Estos contratos son la pieza clave para conseguir la interoperabilidad entre los VLE y las herramientas externas. La arquitectura y sus contratos están pensados para cumplir los requisitos de los actores interesados, tomando para ello compromisos sobre las decisiones de diseño identificadas en el capítulo previo. La versión original de la tesis explica la arquitectura y sus contratos de forma mucho más detallada, discutiendo también la aplicación de esta arquitectura a otros contextos distintos del educativo.

El cuarto capítulo está dedicado a exponer muy brevemente la implementación de referencia que se ha desarrollado para la arquitectura. En él se mencionan los VLE y las herramientas externas que son soportados actualmente, y la idea de que esta implementación de

referencia debe servir de base para aquellos que quieran contribuir a integrar nuevos VLE y herramientas utilizando la arquitectura propuesta. La versión en inglés aporta más detalles sobre la implementación de referencia indicando, entre otras cosas, las tecnologías y el proceso empleados en su construcción. También se proporciona en la versión original información detallada para desarrolladores, administradores y usuarios que deseen utilizar esta implementación de la arquitectura.

El resumen del quinto capítulo se centra en los principales detalles de la evaluación de la arquitectura. Este resumen destaca el cumplimiento de los requisitos de los actores interesados, a partir de evidencias obtenidas de cuatro experiencias con usuarios reales. También se resumen las principales conclusiones de la comparativa llevada a cabo con otros trabajos relacionados que, al igual que la arquitectura propuesta en esta tesis, han sido diseñados para la integración poco acoplada de múltiples herramientas en múltiples VLE. Detalles completos de la metodología de evaluación seguida y de los resultados obtenidos pueden consultarse en la versión en inglés de la tesis.

Para finalizar, se presenta un resumen de las conclusiones de la tesis, recordando los objetivos propuestos al principio de la misma, y también las contribuciones esperadas, indicando que éstas han sido alcanzadas. Antes de dar por terminada la tesis, se esbozan algunas de las líneas de trabajo futuro que surgen de la misma. Como en los capítulos anteriores, la versión extendida de las conclusiones y del trabajo futuro puede encontrarse en la versión en inglés de la tesis.

Índice General

1. Introducción

El uso de las Tecnologías de la Información y las Comunicaciones (TIC) en diferentes áreas, tales como medicina, industria o educación [Abb01] está cambiando la sociedad de una forma que hace algunos años habría sido difícil de pronosticar [Duq05]. La rápida extensión de Internet, las tecnologías web, las redes inalámbricas y los dispositivos móviles han llevado a la adopción de las TIC, no sólo en el trabajo o en la escuela, sino también en nuestra vida cotidiana, en un periodo muy corto de tiempo [Pel01]. Esto abre las puertas a una comunicación ubicua entre las personas a lo largo del mundo, así como también a un acceso omnipresente a muchos servicios y datos [Tho08].

La educación se aprovecha de estas tendencias relacionadas con las TIC en un campo de estudio denominado Aprendizaje Mejorado por la Tecnología (TEL - *Technology Enhanced Learning*) [Joh04], el cual ha sido investigado durante más de dos décadas [Gul08]. TEL estudia cómo apoyar mediante tecnología las actividades de aprendizaje en diferentes niveles, desde la escuela primaria hasta la educación superior [Bat03], y en diferentes contextos, incluyendo aprendizaje formal e informal [Era04,Fol06], pero considerando los desafíos involucrados en el proceso del aprendizaje continuo (*lifelong learning*) [Sha00]. La investigación en TEL también cubre el aprendizaje remoto (a veces llamado también *e-learning*) [Ros11], el aprendizaje presencial y el aprendizaje semipresencial [Osg03], en el cual se combinan actividades tradicionales presenciales y remotas. La importante comunidad de investigadores trabajando en TEL se ejemplifica mediante destacados proyectos, tales como STELLAR[11] (con más de quince socios participando en este proyecto a lo largo de Europa), convocatorias específicas sobre este tema en diversos programas de financiación internacional, como el Séptimo Programa Marco[12], así como numerosas revistas, conferencias y publicaciones recientes en este campo [Bal09,Hak08].

Dentro de TEL, el Aprendizaje Colaborativo Apoyado por Ordenador (CSCL - *Computer Supported Collaborative Learning*) [Kos96] constituye un paradigma multidisciplinar en el cual las TIC se utilizan para facilitar las interacciones sociales y efectivas entre los participantes, y la adquisición de conocimientos y habilidades [Dil99]. Profesionales de la educación con diferente for-

[11] `http://stellarnet.eu`. Última visita: junio 2012.
[12] `http://cordis.europa.eu/fp7/home_en.html`. Última visita: junio 2012.

mación en psicología, educación o tecnología participan normalmente en el desarrollo de sistemas CSCL, los cuales son aplicaciones software que apoyan el diseño, la instanciación, y la puesta en marcha de situaciones de aprendizaje colaborativo [Her06b, Kop05]. Del mismo modo que TEL, CSCL también presenta una importante comunidad de investigadores, como puede comprobarse mediante los principales proyectos activos, tales como Euro-CAT-CSCL[13], un proyecto internacional con socios de tres países diferentes financiado por la Comisión Europea, así como numerosas revistas, conferencias y publicaciones recientes sobre este tema [Bud08, Dil09].

De acuerdo con [Dil99], las situaciones de aprendizaje colaborativo son aquellas que conllevan el aprendizaje mediante la colaboración. Los autores de [Osu99] analizan cinco características que describen las situaciones de aprendizaje colaborativo. La primera de ellas es la *configuración social*, la cual comprende la descripción de los participantes y sus roles (estudiante, educador, administrador, diseñador instruccional, etc.), así como la estructura de grupos, la cual puede ser modificada y actualizada durante la puesta en marcha de la situación de aprendizaje [Mar04]. Los *objetivos* de aprendizaje expresan tanto los objetivos individuales como los grupales que deben ser alcanzados durante la puesta en marcha de la situación de aprendizaje; estos objetivos deben ser formalizados en el diseño de la situación de aprendizaje [Dil02a]. Dicha situación tiene una *estructura*, la cual puede típicamente descomponerse en un conjunto de actividades [Gif99], cada una orientada a la realización de un conjunto de tareas [Dil02a]; esta estructura puede, en ocasiones, organizarse siguiendo un cierto orden de actividades, componiendo así una secuencia de actividades [Dal03]. Aquí es importante apuntar que las situaciones de aprendizaje colaborativo pueden incluir tanto actividades individuales como actividades colaborativas [Osu99, Sta06]. En las actividades colaborativas estructuradas los participantes pueden construir relaciones sólidas para alcanzar un objetivo común, al contrario de lo que ocurre en las actividades colaborativas desestructuradas, en las cuales los participantes no comparten objetivos, y se necesita una mínima dependencia entre ellos [Chi02]. Otra característica que debe definirse es el conjunto de *recursos* que apoyan cada actividad [Her06a]; estos recursos pueden ser herramientas o artefactos [Pal08]. Finalmente, la situación se desarrolla en un *entorno*, en el cual los participantes encuentran la estructura de actividades, herramientas y artefactos, y en el cual los objetivos de aprendizaje y la configuración social pueden ser explícitamente definidos [Bak97]. Con el propósito de conseguir unas interacciones más eficaces, estos entornos CSCL y sus recursos deberían ser personalizados para cada participante, dependiendo de su rol y grupo en cada actividad, y de los objetivos que deban ser alcanzados [Ase08].

Históricamente, la evolución de los entornos CSCL (también denominados sistemas CSCL) comenzó con herramientas y aplicaciones software pequeñas y aisladas, por ejemplo, de correo electrónico, chat o mensajería instantánea, que fueron inicialmente diseñadas para facilitar la comunicación entre usuarios [Sta06]. Estas herramientas se agruparon posteriormente en entornos

[13] `http://cat-cscl.eu`. Última visita: junio 2012.

CSCL, los cuales proporcionaban varias formas de construcción pedagógica para aprendizaje colaborativo, e incluían herramientas compartidas adicionales como calendarios y editores. Como ejemplo, C-CHENE [Bak96] fue uno de los primeros entornos CSCL; su propósito era promover el aprendizaje de conceptos en física a través del modelado, e incluía algunas herramientas colaborativas específicas como un editor de cadenas de energía. Sin embargo, estos primeros sistemas CSCL promovían la colaboración desestructurada entre usuarios, ya que ningún tutor podía definir explícitamente los objetivos de aprendizaje, ni tampoco se permitía formalizar un guión (*script*) con la secuencia de actividades que debía completarse y los objetivos compartidos que debían alcanzarse en cada actividad [Dil02a]. El uso del guiado en situaciones de aprendizaje colaborativo es una práctica común de colaboración estructurada, y se ha demostrado que incrementa la eficacia de las interacciones y el aprendizaje entre estudiantes [Dil02a]. De acuerdo con [Gom09, Her06a], el aprendizaje colaborativo guiado tiene un ciclo de vida compuesto por cuatro fases: la fase de *diseño*, en la cual los educadores (o los diseñadores instruccionales) definen el apoyo computacional, la estructura de actividades, los objetivos de aprendizaje, el número de grupos (pero no los componentes específicos), y las tareas que las herramientas deberían apoyar (p. ej. edición de texto síncrona) para cada actividad [Kop05, Mia05]; la fase de *instanciación*, en la cual los educadores pueblan los grupos, seleccionan las herramientas específicas que se espera utilicen los estudiantes [IMS03], crean las diferentes instancias de herramientas [Bot08, Per10] para cada grupo en cada actividad, y personalizan el entorno, de acuerdo con las necesidades de cada participante; la fase de *puesta en marcha*, en la que los estudiantes realizan las actividades de la situación de aprendizaje colaborativo bajo la monitorización de los educadores (o monitores) [Dil07], los cuales pueden mediar para potenciar el proceso de aprendizaje; y la fase de *evaluación* en la que los educadores (o evaluadores) valoran el conocimiento y las habilidades adquiridas por los estudiantes [Dil02a, Vil09].

La siguiente generación de entornos CSCL permitía la definición de diferentes roles (caracterizando, por ejemplo, a los educadores y a los estudiantes con diferentes permisos en el entorno), y facilitaba la estructuración de situaciones de aprendizaje colaborativo mediante guiones. Algunos ejemplos de entornos CSCL que permitían el uso de roles y el guiado son Universanté [Ber01], un sistema de propósito específico para aprender sobre problemas de salud pública, y Gridcole [Bot05], un sistema de propósito general que podía ser personalizado para ajustarse a las necesidades de los educadores. En ese momento histórico el término Entorno de Aprendizaje Virtual (VLE - *Virtual Learning Environment*) [App99, Dil00] fue acuñado para definir los sistemas CSCL que, como Universanté o Gridcole, permitían definir una jerarquía de roles en la cual el educador era el actor principal, y proporcionaban un entorno compartido personalizable para la realización estructurada de actividades individuales y colaborativas. Sin embargo, todavía no se ha acordado una definición formal de VLE en la comunidad y, en ocasiones, términos como Sistema de Gestión del Aprendizaje (LMS - *Learning Management System*), Sistema de Gestión de Contenidos (CMS - *Content Management System*), Sistema de Gestión de Contenidos de

Aprendizaje (LCMS - *Learning Content Management System*), Entorno de Aprendizaje Gestionado (MLE - *Managed Learning Environment*), o simplemente Plataforma de Aprendizaje (LP - *Learning Platform*), se utilizan como sinónimos de VLE. Curiosamente, algunos autores apuntan diferencias menores entre estos términos. Por ejemplo, los autores de [Dvo11] ven VLE y LMS como nombres intercambiables que se refieren a sistemas software diseñados para facilitar la enseñanza y el aprendizaje utilizando navegadores, y que incluyen herramientas tales como cuestionarios, wikis o blogs, mientras CMS es el nombre empleado para designar a repositorios de datos centralizados.

En esta tesis, un VLE se define como un sistema centrado en el educador que permite el diseño, la instanciación, la puesta en marcha y la evaluación de situaciones de aprendizaje colaborativo mediante un conjunto de actividades síncronas/asíncronas, presenciales/remotas, individuales/colaborativas, las cuales están apoyadas por una colección de recursos y herramientas disponibles; esta definición es consistente con el uso mayoritario del término en la literatura (véase por ejemplo [Dil02b, Sti07, Wel06, Xu05]). Hoy en día, el VLE más comúnmente utilizado es Moodle [14] con más de 66.000 instalaciones en 216 países[15] (en el momento de escritura), aunque otros destacados ejemplos de VLE son LAMS[16], .LRN[17], Sakai[18], Blackboard[19], Claroline[20] o SharePoint LMS[21].

En la última década, los VLE han llegado a ser la corriente dominante, especialmente para el aprendizaje remoto y semipresencial, tanto en academia [Dun03, Wel06] como en industria [Mor03]. Sin embargo, algunos profesionales consideran que los VLE están muy centrados en cubrir las necesidades de las instituciones, en lugar de las de los propios estudiantes [Sev08]. Por ello, una tendencia de investigación que propone alternativas software centradas en el estudiante, las cuales pueden ser agrupadas bajo el término Entorno de Aprendizaje Personal (PLE - *Personal Learning Environment*) [Att07, Har06, Wil06], ha surgido con fuerza en los últimos años. A pesar de que el término es relativamente nuevo, algunos trabajos de investigación en este tema, como Symba [Bet03], el cual promueve la personalización del entorno por los propios estudiantes, fueron publicados hace casi una década. Por el contrario, algunos profesionales todavía consideran que los educadores deben ser responsables de proporcionar a los estudiantes los recursos de aprendizaje adecuados para desarrollar los conocimientos y las habilidades esperadas de forma sistemática [Mue11, Wel07a], como sucede en los VLE. Sin embargo, todos ellos están de acuerdo en que los PLE no reemplazarán a los VLE, ya que ambos pueden coexitir, e incluso fusionarse,

[14] `http://moodle.org`. Última visita: junio 2012.
[15] `http://moodle.org/sites`. Última visita: junio 2012.
[16] `http://lamsinternational.com`. Última visita: junio 2012.
[17] `http://openacs.org/projects/dotlrn`. Última visita: junio 2012.
[18] `http://sakaiproject.org`. Última visita: junio 2012.
[19] `http://blackboard.com`. Última visita: junio 2012.
[20] `http://claroline.net`. Última visita: junio 2012.
[21] `http://sharepointlms.com`. Última visita: junio 2012.

dependiendo de los escenarios de aprendizaje y de los objetivos perseguidos. Por ejemplo, los autores de [Wil06] sugieren que los PLE serán dominantes en aprendizaje informal y basado en competencias, mientras que los VLE quedarán reservados para la educación formal. En este punto, es importante notar que esta tesis centra su ámbito de investigación en los VLE, pero que las contribuciones esperadas, que serán más tarde definidas, podrían aplicarse a los VLE, así como a otros entornos que puedan ser empleados para el aprendizaje colaborativo, tales como wikis [Aug04] o redes sociales [Lip02].

Problema de investigación de la tesis

Los VLE son actualmente uno de los sistemas más extendidos para el apoyo a las situaciones de aprendizaje colaborativo [Wel07b]. La mayoría de VLE, como por ejemplo Moodle, LAMS, Sakai o Blackboard incluyen las características descritas en [Osu99] para fomentar el aprendizaje mediante la colaboración. En este sentido, los VLE permiten la definición de una configuración social de participantes basada en grupos y roles, su estructuración en actividades, cursos y/o lecciones con objetivos de aprendizaje predefinidos, y el uso de varias herramientas y recursos en cada actividad. Además, los VLE dan típicamente apoyo a las cuatro fases originalmente definidas para el ciclo de vida del aprendizaje colaborativo guiado, aunque sólo algunos de ellos, como LAMS, permiten la formalización de guiones. Por tanto, los educadores pueden diseñar e instanciar actividades de aprendizaje individuales y colaborativas en los VLE que los estudiantes pueden poner en marcha posteriormente, siendo su conocimiento adquirido evaluado por los educadores. La implementación de este ciclo de vida depende de la filosofía y la arquitectura del VLE. Por ejemplo, Moodle no separa explícitamente diseño, instanciación y puesta en marcha, ya que está basado en la filosofía del bricolaje pedagógico, permitiendo a los educadores refinar e iterar sobre el diseño de aprendizaje a medida que las actividades se van realizando [Ber05]. Por otro lado, LAMS separa explícitamente diseño, instanciación y puesta en marcha en tres entornos diferentes: autoría, monitorización, y aprendizaje [Dal03]. Sin embargo, existe un cierto solapamiento en la definición de estas fases en LAMS [Gom09,Her06a], debido a que parte de la instanciación (la selección de herramientas) se lleva a cabo en el entorno de autoría.

Aquellos profesionales de la educación que diseñan e instancian situaciones de aprendizaje colaborativo con la mediación de los VLE deben seleccionar el conjunto de herramientas que quieren que los estudiantes utilicen para completar las actividades que componen la situación de aprendizaje. Los VLE normalmente incluyen un limitado conjunto de entre diez y veinte herramientas nativas para propósitos individuales y colaborativos, que pueden ser añadidas a cada actividad. Algunos ejemplos de herramientas que aparecen en los principales VLE son chats, foros, tablones de noticias, cuestionarios y encuestas [Col07, Dal03, Uzu06], aunque la implementación y funcionalidad ofrecida por cada una de esas herramientas depende del VLE

concreto; por ejemplo la implementación del chat de Moodle es diferente a la del chat de LAMS. Estas herramientas nativas se diseñaron en su mayoría para tareas de propósito específico, y por tanto, pueden utilizarse en un variado conjunto de actividades de aprendizaje. A pesar de ello, el conjunto reducido de herramientas nativas ha sido criticado frecuentemente, considerándolo una importante limitación para el apoyo de actividades de aprendizaje [Bow11, Dag07, Fie07, Liv08].

El problema del conjunto reducido de herramientas disponibles en las plataformas de aprendizaje puede parecer reciente, aunque no es el caso. De hecho, tres alternativas diferentes se han propuesto históricamente para superar este problema. Las investigaciones pioneras tenían como objetivo diseñar y desarrollar nuevos VLE flexibles y extensibles que estuvieran específicamente diseñados para facilitar la incorporación de herramientas externas. Este es el caso de DARE [Bou01], Symba [Bet03] o Gridcole [Bot05, Bot08], y más recientemente Pelican [Vel09]. Sin embargo, la principal limitación de estos VLE es que habían sido concebidos para reemplazar otros VLE, y aquellos educadores y estudiantes que ya estaban acostumbrados a uno diferente en sus clases eran reticentes a adoptar estos nuevos VLE, a veces por el esfuerzo de aprendizaje y el tiempo de adaptación requeridos, y otras veces porque las propias instituciones les obligaban a utilizar un VLE concreto [Ala10a]. Además, los mencionados VLE eran productos de proyectos de investigación que nunca llegaron a ser productos estables, dificultando así su adopción por aquellas instituciones potencialmente interesadas. Algunos otros autores decidieron desarrollar desde cero herramientas para ciertos VLE. Este es el caso de muchos módulos y *plugins* para Moodle[22] [Gut09] o de los Bloques de Construcción para Blackboard (*Blackboard Building Blocks*) [Pit03], los cuales extendían el conjunto de herramientas disponibles en Moodle y Blackboard, respectivamente. Estas herramientas podían proporcionar la funcionalidad esperada por algunos educadores, combinada con las características propias de los VLE y, en algunos casos, fueron exitosamente adoptadas, e incluso empaquetadas en las sucesivas versiones oficiales de estos VLE. Sin embargo, los educadores que usaban otros VLE, como LAMS o Sakai, no podían incluir estas herramientas para apoyar sus actividades de aprendizaje. Además, estas herramientas podían estar también reemplazando a otras herramientas existentes similares, y por tanto, de nuevo, los educadores y estudiantes debían asumir un esfuerzo de aprendizaje y un periodo de adaptación extra. El resultado de estas dos alternativas fue una falta de adopción de los trabajos relacionados, provocando el auge de una tercera opción: **la integración de herramientas externas existentes en VLE existentes** [Fon09, Fue11, Sev08].

La integración de herramientas existentes en VLE existentes tiene como objetivo ofrecer a los educadores un conjunto más extenso de herramientas disponibles en sus VLE habituales para apoyar sus actividades de aprendizaje [Fue11]. Los investigadores y desarrolladores trabajando en esta línea se han visto favorecidos por la reciente extensión de las tecnologías web [Pau08] y el auge de la Web 2.0 [ORe07], los cuales han supuesto una explosión de herramientas software de terceros

[22] `http://moodle.org/plugins`. Última visita: junio 2012.

utilizadas cada vez más por los profesionales de la educación, en principio, fuera de los VLE [Wel07b]. Además, la mayoría de estas herramientas, como por ejemplo Google Apps[23], Twitter[24], Wordpress[25], Flickr[26] o Doodle[27], están disponibles de forma gratuita para educación (y algunas de ellas para cualquier otro uso), lo que las convierte en alternativas muy interesantes para escuelas, colegios o universidades que no pueden permitirse pagar múltiples licencias software por aplicaciones comerciales. Una clara señal del éxito de las herramientas software en educación es la publicación por parte de algunos centros educativos de listas con las herramientas más utilizadas por los educadores en sus clases. Este es el caso del Centro de Tecnologías de Aprendizaje y Rendimiento (C4LPT - *Centre for Learning & Performance Technologies*) y su lista con las 100 mejores herramientas para educación (*Top 100 Tools for Learning*)[28] que se actualiza cada año con las preferencias más destacadas de los educadores en lo que respecta a aplicaciones, sitios web, plataformas de aprendizaje y dispositivos hardware. Como conclusión, el enorme número de herramientas disponibles para educación, la popularidad de los VLE, y sus limitaciones con respecto a las herramientas que ofrecen nativamente, han motivado que un significativo número de recientes trabajos de investigación estén abordando la integración de herramientas existentes en VLE existentes [Bol07, Boo09, Bla09, Dod08, Fon09, Fue11, IMS06c, Sev08].

Sin embargo, la integración de una herramienta en un VLE no es una tarea sencilla, principalmente por dos razones. Primero, cada VLE y cada herramienta típicamente impone diferentes requisitos heterogéneos para permitir su extensión funcional e interoperabilidad tecnológica con otros sistemas. Estos requisitos se definen en los contratos de integración [Ghi06, Lar02], sean éstos explícitos o no. Un contrato de integración determina, al menos, las tecnologías, las interfaces, y los modelos de datos que deben emplearse para permitir la comunicación entre un sistema y otras aplicaciones [Ala12a]. Generalmente, cuantos más requisitos se definan en estos contratos, más acoplada es la integración, y más rica es la comunicación que puede establecerse entre los VLE y las herramientas, al contrario de lo que ocurre en aquellos contratos con pocos requisitos que promueven el bajo acoplamiento, a cambio de una comunicación más pobre entre los VLE y las herramientas [Pau09]. En segundo lugar, un desarrollador debe programar el código necesario para permitir la comunicación entre un contrato de VLE y un contrato de herramienta, completando así la integración de una herramienta en un VLE. El papel de desarrollador puede desempeñarlo cualquiera que esté interesado en esa integración; esto incluye al proveedor de VLE, al proveedor de herramienta, o a un tercero [Ala10a]. Los desarrolladores que asumen este esfuerzo normalmente esperan un beneficio a cambio, el cual puede ser reconocimiento, reputación, compensaciones económicas, o la satisfacción de usar (o de dejar que otros usen) las herramientas

[23] `http://google.com/apps/intl/en/edu`. Última visita: junio 2012.
[24] `http://twitter.com`. Última visita: junio 2012.
[25] `http://wordpress.org`. Última visita: junio 2012.
[26] `http://flickr.com`. Última visita: junio 2012.
[27] `http://doodle.com`. Última visita: junio 2012.
[28] `http://c4lpt.co.uk/recommended/2011.html`. Última visita: junio 2012.

integradas. Los trabajos de investigaciones que tratan la integración de herramientas externas en VLE deben considerar en sus propuestas los desarrolladores esperados; si no, podría suceder que nadie quisiera desarrollar el código necesario.

El desarrollo del código que permite la interoperabilidad entre los diferentes contratos de integración involucra un cierto **esfuerzo de desarrollo** [Alb83]. Sin embargo, este esfuerzo de desarrollo es significativamente alto en la mayoría de aproximaciones de integración, lo que se debe principalmente a dos factores. Primero, muchas aproximaciones promueven una integración uno a uno (*one-to-one*) entre los VLE y las herramientas. Esto implica que siempre debe desarrollarse código nuevo para cada integración, como ocurre con la mayoría de *plugins* para Moodle. Por ejemplo, si un desarrollador asume el esfuerzo de integrar Flickr en Moodle utilizando su propio mecanismo de extensión, prácticamente no podrá reutilizar nada del código generado, ni del conocimiento adquirido, al integrar Flickr en LAMS, Sakai o Blackboard. Segundo, muchas aproximaciones promueven una integración acoplada [Ort90] entre los VLE y las herramientas. Esto requiere generar una cantidad significativa de código extra dirigido a permitir interacciones más ricas entre ellos, incluso aunque estas interacciones no sean necesarias para llevar a cabo la mayoría de situaciones de aprendizaje. La especificación IMS *Learning Tools Interoperability* (IMS LTI - Interoperabilidad de Herramientas de Aprendizaje) [IMS06c] (popularmente llamada *Full* LTI) y el Servicio de Integración de CopperCore (CCSI - *CopperCore Service Integration*) [Vog06] son dos ejemplos de propuestas de integración que conllevan un alto esfuerzo de desarrollo debido a este segundo factor. La lección que puede aprenderse de estas y otras propuestas similares es que un alto esfuerzo de desarrollo limita la adopción de las aproximaciones de integración, ya que puede desanimar a los desarrolladores a contribuir a la integración de nuevas herramientas y VLE, reduciendo por tanto, el interés de los profesionales de la educación y de las instituciones en estas aproximaciones.

Con el objetivo de abordar estos problemas, se han propuesto recientemente dos trabajos que tratan de reducir el esfuerzo de desarrollo, fomentando una integración muchos a muchos (*many-to-many*) entre VLE y herramientas, y siguiendo una aproximación de bajo acoplamiento. Es el caso de Apache Wookie (*Incubating*)[29], una implementación de la arquitectura propuesta en [Wil08], que permite la integración de pequeñas aplicaciones, siempre y cuando hayan sido desarrolladas siguiendo la especificación W3C para *widgets* [W3C11]. Sin embargo, esta es una **restricción tecnológica muy estricta** que dificulta la integración de muchas otras herramientas existentes que no la cumplen (p. ej. Google Apps), aunque puedan ser de interés para apoyar muchas actividades de aprendizaje. Por tanto, esta restricción reduce el conjunto de herramientas externas que los educadores podrían integrar en sus situaciones de aprendizaje mediante Apache Wookie. Además, la especificación W3C para *widgets* ha sido definida para estandarizar herramientas simples o *mash-ups*, reduciendo por tanto la funcionalidad de las herramientas

[29] `http://incubator.apache.org/wookie`. Última visita: junio 2012.

disponibles que los educadores podrían utilizar en sus situaciones de aprendizaje. En este caso, la conclusión es que la imposición de requisitos tecnológicos estrictos, como los de Apache Wookie, puede desanimar a las instituciones y a los profesionales a la hora de tomar la decisión sobre la adopción de esta y otras aproximaciones con una limitación similar.

Otro trabajo que también ha sido propuesto recientemente con el objetivo de reducir el esfuerzo de desarrollo, fomentando la integración muchos a muchos y siguiendo una aproximación de bajo acoplamiento, es IMS Basic LTI (*Basic Learning Tools Interoperability* - Interoperabilidad de Herramientas de Aprendizaje Básica) [IMS10b], un subconjunto de la mencionada Full LTI [IMS06c]. Basic LTI, el cual se ha fusionado este año junto con Full LTI en una especificación única llamada simplemente LTI [IMS12], permite la integración sencilla de un conjunto amplio de herramientas existentes en VLE. Sin embargo, Basic LTI también presenta una importante limitación: no se responsabiliza de la gestión del acceso de usuarios y grupos a instancias de herramientas, ya que sólo permite recuperar una única instancia genérica para cada herramienta externa. Como consecuencia, Basic LTI no permite a los educadores solicitar la creación y configuración de instancias separadas para cada grupo definido en las actividades de aprendizaje desde los VLE, no siendo responsable de la gestión del acceso de grupos y usuarios a la funcionalidad y contenido de las instancias de herramientas externas. Por tanto, Basic LTI no puede hacer uso de las propiedades colaborativas proporcionadas por los VLE, tales como el uso de grupos, para establecer la configuración social durante la instanciación de situaciones de aprendizaje colaborativo [Mar04,Osu99]; ni tampoco permite la personalización de las herramientas integradas, para favorecer la aparición de interacciones más eficaces entre los estudiantes durante la puesta en marcha de estas situaciones [Ase08]. En otras palabras, Basic LTI es una aproximación muy limitada para **permitir la instanciación y puesta en marcha de situaciones de aprendizaje colaborativo**. Esta limitación puede motivar que muchos profesionales e instituciones que promueven la colaboración entre los estudiantes en sus prácticas puedan descartar Basic LTI u otras aproximaciones similares que también presenten esta limitación.

Con todo ello, los trabajos de integración existentes presentan tres importantes limitaciones que dificultan su adopción generalizada para el apoyo a las situaciones de aprendizaje colaborativo. Una solución que podría superar estas limitaciones sería la definición e implementación de una arquitectura *middleware* [Bri04] que: promueva la integración muchos a muchos y el bajo acoplamiento para reducir el esfuerzo de desarrollo; imponga únicamente restricciones que la mayoría de proveedores de VLE y herramientas ya cumplan; y facilite la instanciación y puesta en marcha de actividades individuales y colaborativas que requieran la integración de herramientas externas, apoyando la creación y configuración de instancias de estas herramientas, de acuerdo con la configuración social definida en los VLE. Además, esta arquitectura debería ser compatible con otras aproximaciones de integración de bajo acoplamiento, de tal forma que la adopción de una de ellas no impida la utilización de las otras.

Objetivos y contribuciones

En la sección previa se han descrito las tres principales limitaciones de los trabajos de investigación actuales que tratan la integración de herramientas externas en VLE. Estas limitaciones son las mayores responsables de la falta de adopción generalizada de las propuestas de integración existentes. Para superar estas limitaciones, el objetivo global de esta tesis es:

Diseñar, desarrollar y evaluar una arquitectura *middleware* que permita la integración de múltiples herramientas externas existentes en múltiples VLE existentes, requiriendo un esfuerzo de desarrollo asumible para integrar nuevas herramientas y VLE, imponiendo sólo restricciones básicas que la mayoría de proveedores de VLE y herramientas cumplan, y ofreciendo suficiente funcionalidad para facilitar la instanciación y puesta en marcha de situaciones de aprendizaje colaborativo.

Para cumplir este objetivo global, se establecen varios objetivos parciales con sus correspondientes contribuciones:

- *Analizar el problema de la integración de herramientas externas existentes en VLE existentes.*

  Las contribuciones de la tesis a este objetivo son: la **identificación de los requisitos de los principales actores interesados** y la **identificación de las principales decisiones de diseño y alternativas a la hora de integrar herramientas externas en VLE**. Tanto los requisitos de los actores interesados como las alternativas a estas decisiones guían el diseño y desarrollo de la arquitectura. Las conclusiones sobre este objetivo parcial y las contribuciones han sido publicadas en [Ala10a], incluyendo la definición de las principales decisiones de diseño y las propuestas de integración hasta 2009, siendo revisadas y actualizadas en [Ala10c] para incluir trabajos de integración posteriores. Además, los requisitos de los actores interesados se han plasmado formalmente en [Ala12a].

- *Proponer una arquitectura middleware que permita la integración de múltiples herramientas externas existentes en múltiples VLE existentes y que supere las limitaciones de los trabajos relacionados.*

  Teniendo en cuenta las limitaciones de los trabajos relacionados, se distinguen tres partes en la propuesta de la arquitectura:

  - Definir y detallar las *restricciones a imponer sobre VLE y herramientas*, teniendo en cuenta que deben ser cumplidas por la mayoría de proveedores.
  - Definir y detallar el *contrato de integración de la arquitectura*, teniendo en cuenta que debe reducir el esfuerzo de desarrollo necesario para integrar herramientas existentes en VLE existentes.

- Definir y detallar los *elementos de la arquitectura y sus responsabilidades*, considerando que deben facilitar la instanciación y puesta en marcha de actividades de aprendizaje colaborativo que requieran la integración de herramientas externas.

La contribución de la tesis a este objetivo es **GLUE! (*Group Learning Uniform Environment* - Entorno Uniforme de Aprendizaje en Grupo), una arquitectura *middleware* que cumple con los objetivos deseados**, superando por tanto, las limitaciones de los trabajos relacionados. La propuesta de esta arquitectura se detalla en [Ala12a].

- *Desarrollar una implementación de referencia de la arquitectura propuesta.*

  La contribución de la tesis a este objetivo parcial es **GLUE!-RI (*GLUE! Reference Implementation* - Implementación de referencia de GLUE!)**. Esta implementación y su prototipo han sido publicados en [Ala12a], y con más detalles para la particularización de LAMS en [Ala11b]. Además, [Ala12c] es un artículo de demostración de la utilización de la implementación de referencia en Moodle y LAMS. El código de GLUE!-RI está disponible para su descarga e instalación en `http://gsic.uva.es/glue`.

- *Evaluar la arquitectura y su implementación de referencia.* Esta evaluación debe considerar que los objetivos y contribuciones de esta tesis pertenecen al campo del CSCL y al de la integración software y que, por tanto, las dimensiones educativa y tecnológica del problema deben abordarse en la evaluación [Zel02]. Con estas consideraciones, la evaluación comprende:

  - *Cuatro experiencias con educadores y estudiantes reales* [Dew01] en cursos de educación superior.
  - *Una comparación de GLUE! y otras aproximaciones de bajo acoplamiento.*

  Esta evaluación ha sido publicada parcialmente en [Ala12a], mostrando los resultados y las conclusiones de las cuatro experiencias, así como la comparativa parcial de las aproximaciones de bajo acoplamiento. Además, la evaluación extendida centrada en dos de las experiencias puede consultarse en [Ala12b].

Metodología de investigación

Esta tesis se enmarca dentro de un campo multidisciplinar, ya que sus objetivos y contribuciones están orientados al diseño y desarrollo de un sistema software, que se espera tenga un cierto impacto en el dominio educativo. Por este motivo, la metodología de investigación debería también combinar diferentes métodos para tener en cuenta la naturaleza multidisciplinar de la investigación. En este sentido, el método de ingeniería de Adrion [Adr93] podría cubrir la

mayor parte de la metodología necesaria para esta tesis, iterando en los siguientes cuatro pasos: observar las soluciones existentes, proponer una mejor solución, construir o desarrollar, medir y analizar. Sin embargo, este método podría combinarse, especialmente en la parte de evaluación, con el método empírico [Big82], debido al uso de experiencias [Dew01] como parte de la evaluación educativa. Glass [Gla95] formaliza la combinación de estos dos métodos, tomando el de Adrion como referencia, en cuatro fases iterativas que ya han sido aplicadas a tesis recientes con contribuciones tecnológicas en el campo del CSCL [Her07b, Per11, Vil10]:

- **Fase informativa**. La primera fase tiene como objetivo la recogida de información sobre el dominio en el que se lleva a cabo la investigación. Para ello, la primera tarea es la revisión y análisis de la literatura existente y los trabajos previos para detectar posibles preguntas de investigación. A continuación, es conveniente participar en eventos y proyectos científicos para valorar el potencial interés de esas preguntas de investigación y obtener realimentación de la comunidad de investigadores en este campo. En esta tesis, esta fase incluye la revisión de múltiples trabajos en TEL, CSCL, plataformas de aprendizaje, uso de herramientas para propósitos educativos, tendencias tecnológicas, ingeniería de software, arquitecturas software e integración software en múltiples contextos. Durante este proceso, la integración de herramientas externas en VLE para facilitar la realización de un amplio rango de situaciones de aprendizaje colaborativo surge como la principal pregunta de investigación para la tesis. Además, esta fase también incluye la participación en proyectos europeos, nacionales y regionales relacionados con el dominio, la asistencia a varias conferencias, y la realización de estancias cortas con expertos en integración software y plataformas de aprendizaje.

- **Fase proposicional**. La segunda fase tiene como objetivo proponer o formular hipótesis o soluciones a las preguntas de investigación identificadas en la fase informativa. En este sentido, la revisión de la literatura permite la identificación de las limitaciones de los trabajos relacionados y el surgimiento de nuevas alternativas originales para abordar estas limitaciones. En esta tesis, esta fase comprende la identificación de los requisitos de los principales actores interesados, el análisis de las principales decisiones de diseño que deberían considerarse al enfrentarse al problema de integración, y la propuesta de una arquitectura de bajo acoplamiento que tiene en cuenta estos requisitos y decisiones de diseño para superar las limitaciones de aproximaciones previas.

- **Fase analítica**. La tercera fase tiene como objetivo analizar y explorar las proposiciones hechas en la fase previa para conseguir una demostración o formulación de principios. Esto puede incluir el desarrollo de los sistemas o aplicaciones necesarios para facilitar esta demostración. En el contexto de esta tesis, se desarrolla una implementación de referencia de la arquitectura propuesta en la segunda fase, proporcionando así un prototipo que puede ser probado y usado por profesionales. Esta implementación permite el análisis y evaluación

de las contribuciones de la tesis, y puede ser mejorada a medida que las iteraciones a lo largo de las fases de esta metodología se vayan sucediendo.

- **Fase evaluativa**. La última fase tiene como objetivo evaluar las proposiciones por medio de la experimentación o la observación, con la ayuda de los sistemas o aplicaciones desarrollados en la fase analítica (si se aplica al caso). En esta tesis, la arquitectura se evalúa por medio de cuatro experiencias reales que tienen como objetivo demostrar si los requisitos de los principales interesados se cumplen, y si se superan las limitaciones de los trabajos previos. Una comparación con otras propuestas de bajo acoplamiento también se lleva a cabo dentro de esta fase.

2. Integración de herramientas externas en VLE

El aprendizaje colaborativo es un proceso en el que el conocimiento se construye a través de las interacciones con otros compañeros y, en la mayoría de los casos, tiene como resultado un aprendizaje más eficaz, en comparación con los procesos pedagógicos individuales o los competitivos [Kos96]. El uso de tecnología para fomentar estas interacciones es abordado de forma conjunta por profesionales de la educación, psicólogos y tecnólogos en el campo de investigación denominado CSCL. Los VLE son sistemas que se han utilizado de forma destacada para el apoyo a la educación y a la colaboración en todos los niveles educativos durante los últimos años [Wel07b], ya que incluyen características colaborativas y pedagógicas significativas, tales como la gestión de grupos, roles, actividades y cursos. Los profesionales de la educación utilizan los VLE de forma recurrente para el diseño, instanciación, puesta en marcha y evaluación de situaciones de aprendizaje colaborativo. Sin embargo, el conjunto limitado de herramientas nativas de los VLE dificulta, e incluso impide, la realización de muchas actividades de aprendizaje [Con10,Dag07,Liv08]. En este punto, la popularidad creciente de las tecnologías y servicios web [Pap03] ha causado un auge en el interés investigador hacia la integración de herramientas externas en VLE, con el objetivo de permitir la realización de un mayor número de situaciones de aprendizaje colaborativo. Esta línea de investigación reemplaza a tendencias anteriores, de acuerdo con las cuales, el número de herramientas nativas de los VLE era aumentado por medio del desarrollo de nuevos VLE y herramientas desde cero.

Sin embargo, la integración de herramientas externas en VLE es un problema muy complejo, principalmente debido a la gran variedad de contratos de integración heterogéneos existentes [Ghi06], y ninguna de las aproximaciones actuales en este tema ofrece una solución genérica con una adopción ampliamente extendida. De esta forma, se justifica la necesidad de plantear una nueva aproximación para este problema. En este punto, se estudia el contexto que rodea al problema, identificando los requisitos de los principales actores interesados, así como las decisiones de diseño y alternativas tomadas por otros trabajos de integración.

Requisitos de los principales actores interesados

Los *profesionales de la educación*, tanto los educadores como los estudiantes, son los actores interesados que utilizan las herramientas externas. A ellos les gustaría emplear las herramientas integradas de la misma forma que utilizan las herramientas nativas. Esto incluye beneficiarse de las principales características de los VLE, entre las cuales destacan el apoyo a la colaboración y al trabajo en grupo [Bow11]. Por ello, las nuevas aproximaciones de integración deberían **permitir la instanciación de actividades individuales y colaborativas que requieran la integración de herramientas externas con un esfuerzo asumible para los educadores (REQ1)**; y **permitir la puesta en marcha de estas actividades facilitando la colaboración entre los participantes (REQ2)**. Además, a los educadores y estudiantes no les gustaría tener que dejar el VLE y las herramientas a los que ya están acostumbrados. Por ello, las nuevas aproximaciones deberían **permitir la integración de VLE y herramientas existentes y populares (REQ3)**. Finalmente, para dar apoyo al mayor número de actividades posible, estas aproximaciones deberían **permitir la integración de muchas herramientas externas (REQ4)**.

Los *desarrolladores* escriben el código necesario para la integración de nuevas herramientas y VLE. Los desarrolladores generalmente se comprometen a escribir ese código con una menor probabilidad si deben asumir un esfuerzo de desarrollo alto. Por tanto, las nuevas aproximaciones deberían **demandar un esfuerzo de desarrollo asumible para la integración de nuevos VLE y herramientas (REQ5)**.

Finalmente, los *proveedores de VLE y herramientas* raramente quieren modificar sus sistemas para cumplir con una nueva aproximación de integración. Frecuentemente, desaprueban también que otros modifiquen sus sistemas, ya que eso puede causar incompatibilidades con las versiones oficiales de VLE y herramientas. Por tanto, las nuevas aproximaciones de integración deberían **ser construidas sobre los VLE y las herramientas (REQ6)**, en lugar de modificar sus implementaciones.

Decisiones de diseño y alternativas

Las nuevas aproximaciones de integración deberían tener en cuenta varias decisiones de diseño técnicas y funcionales, y sus alternativas. Sin embargo, varias de estas decisiones están interrelacionados por lo que necesitan que se establezcan compromisos y prioridades.

Una decisión técnica importante es el **número de restricciones impuestas a VLE y herramientas**. Ejemplos de estas restricciones son lenguajes de programación o modelos de

intercambio de datos. Imponer muchas restricciones puede excluir herramientas y VLE interesantes. Imponer pocas restricciones reduce generalmente el esfuerzo de desarrollo y, por tanto, las posibilidades de que algunas herramientas y VLE populares sean descartados.

Otra decisión relacionada es el **grado de adopción de las restricciones**. Cuanto más extendidas sean las restricciones impuestas, más herramientas y VLE las cumplirán nativamente, y como consecuencia, estas herramientas y VLE serán integrados típicamente con menor esfuerzo y sin modificar su implementación original. Por el contrario, las aproximaciones *ad hoc* pueden imponer restricciones con menor adopción, ya que su objetivo habitual es fomentar la integración de una herramienta específica en un VLE específico.

Otra decisión importante es la **multiplicidad de la integración**. Las aproximaciones genéricas están diseñadas para promover la integración de múltiples herramientas en múltiples VLE (integración muchos a muchos). Estas aproximaciones pueden reducir el esfuerzo de desarrollo al facilitar la reutilización de código, pero con la consecuencia de permitir sólo características y comportamientos limitados, debido a la heterogeneidad existente en lo que respecta a VLE y herramientas. Las aproximaciones *ad hoc* fomentan típicamente la integración uno a uno, permitiendo interacciones más ricas entre VLE y herramientas, aunque sólo sirven para un caso concreto. De forma alternativa, otras aproximaciones pueden facilitar la integración de múltiples herramientas en un VLE (uno a muchos) y, con menor frecuencia, la integración de una herramienta en varios VLE (muchos a uno).

El **grado de acoplamiento** es otra decisión técnica importante relacionada con las restricciones y la multiplicidad. Un acoplamiento fuerte (alto acoplamiento) permite controlar el comportamiento de la herramienta desde el VLE en mayor medida, pero a cambio de un esfuerzo de desarrollo extra. Un acoplamiento débil (bajo acoplamiento) permite un menor control sobre las herramientas integradas, pero también sirve para reducir el esfuerzo de desarrollo.

El **grado de funcionalidad ofrecido** por una aproximación de integración es una importante decisión funcional relacionada con todas las decisiones técnicas presentadas. Ofrecer un alto grado de funcionalidad normalmente requiere imponer más restricciones, un alto acoplamiento y, posiblemente, una integración uno a uno. Por otro lado, ofrecer un bajo grado de funcionalidad generalmente necesita menos esfuerzo de desarrollo y típicamente menos restricciones sobre VLE y herramientas.

Además, en este contexto particular, la **gestión del ciclo de vida de las herramientas externas** [Gom09] es una importante decisión funcional dentro del grado de funcionalidad ofrecido, ya que la mayoría de VLE (y muchas de las herramientas) han sido diseñados para apoyar actividades de aprendizaje individuales y colaborativas. Esta decisión se refiere a la creación, configuración y asignación de diferentes instancias de herramientas para cada usuario o grupo en una actividad de aprendizaje, lo cual puede ser una carga importante si existen flujos de colaboración complejos con muchos grupos y herramientas. Las aproximaciones que permiten gestionar

este ciclo de vida facilitan la instanciación y puesta en marcha de las situaciones de aprendizaje colaborativo. Por el contrario, las que no tienen en cuanta esta decisión de diseño, dificultan los procesos de instanciación y puesta en marcha de estas situaciones de aprendizaje, a pesar de su interés educativo.

Conclusiones

Las diferentes propuestas de integración analizadas con detalle en la versión en inglés de este documento han tomado diferentes alternativas sobre las decisiones de diseño. Algunas de estas alternativas han tenido como consecuencia importantes limitaciones, las cuales pueden considerarse lecciones aprendidas para otros investigadores que tratan el problema de la integración multiplataforma de herramientas de terceros en el dominio educativo, o en otros dominios. Para superar estas limitaciones el siguiente capítulo presenta la contribución central de la tesis, una arquitectura que satisface los requisitos de los principales actores y que se caracteriza por: **un bajo número de restricciones ampliamente adoptadas**, facilitando así la integración de herramientas y VLE populares sin modificar su código; una **integración muchos a muchos y poco acoplada**, promoviendo la integración de muchos VLE y herramientas con un menor esfuerzo de desarrollo; y un **grado de funcionalidad suficiente** que permite, al menos, **la gestión del ciclo de vida de las herramientas externas**. Estas decisiones funcionales son el resultado del compromiso entre los requisitos de los actores interesados, y el resto de decisiones tecnológicas.

3. La arquitectura GLUE!

La arquitectura GLUE! se diseña para permitir la integración de múltiples herramientas externas existentes en múltiples VLE existentes, imponiendo pocas restricciones que los principales proveedores de VLE y herramientas cumplen, reduciendo el esfuerzo de desarrollo necesario para integrar nuevos VLE y herramientas, y permitiendo la gestión del ciclo de vida de las herramientas externas.

Visión general de la arquitectura

La Figura 1 presenta la visión general de la arquitectura GLUE!. GLUE! sigue una *arquitectura de tres capas* compuesta por **servicios distribuidos poco acoplados**, donde diferentes VLE y herramientas se comunican mediante una capa intermedia de software y un conjunto de adaptadores. En los extremos izquierdo y derecho de la arquitectura se utiliza el conocido **patrón adaptador** [Gam95] para envolver los VLE y las herramientas, adaptando sus contratos heterogéneos específicos a dos nuevos contratos homogéneos intermedios: el **contrato de GLUE! para VLE** y el **contrato de GLUE! para herramientas**. Este tipo de adaptadores, que también han sido utilizados en algunas arquitecturas de dos capas [IMS10b,Fue11] para permitir la integración de herramientas en VLE sin modificar su código, se denominan **adaptadores de VLE** y **adaptadores de herramientas** en la arquitectura GLUE!.

La arquitectura también incluye una capa intermedia de software denominada **núcleo de GLUE!**, la cual ofrece los contratos de integración de GLUE!. Este núcleo de la arquitectura, compuesto por un elemento de procesamiento (**GLUElet Manager**) y un elemento de almacenamiento (**registro interno de herramientas**), desacopla los adaptadores de VLE y de herramientas, y asume buena parte de la funcionalidad de integración. Esto facilita la implementación independiente de los adaptadores y reduce su esfuerzo de desarrollo. Con ello, la arquitectura GLUE! fomenta una **integración muchos a muchos**, debido a que cada nuevo adaptador de herramienta desarrollado para una herramienta concreta (o para varias herramientas) permite su integración en cualquier VLE con el correspondiente adaptador de VLE, y viceversa.

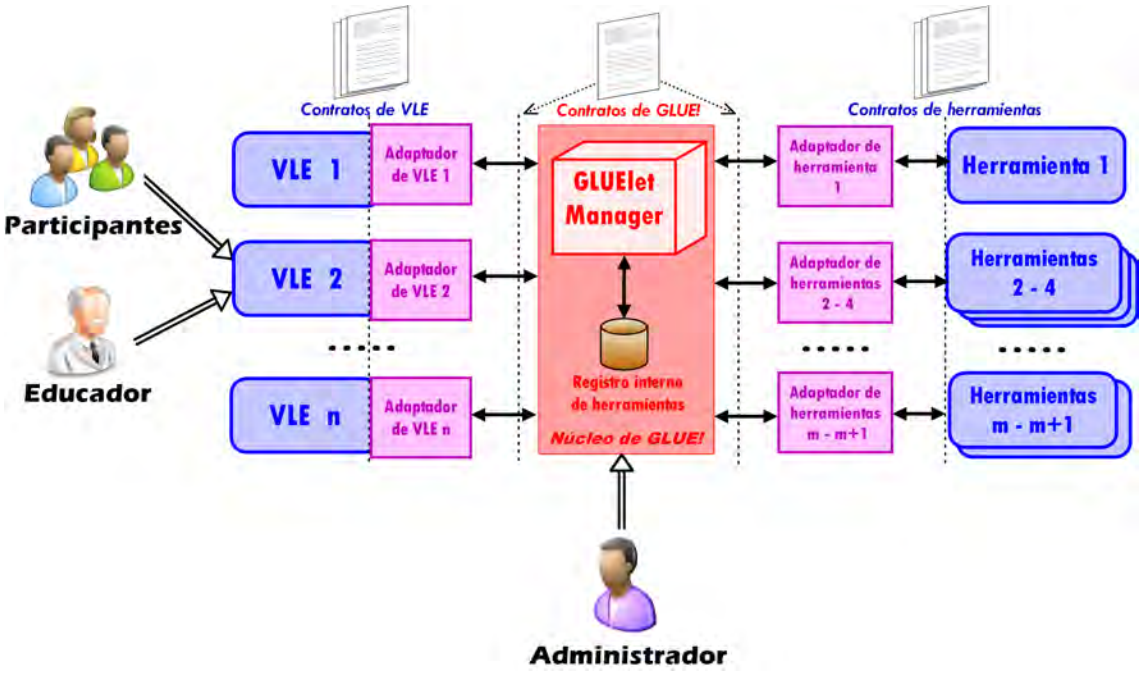

Figura 1: Visión general de la arquitectura GLUE!.

La arquitectura GLUE! proporciona la funcionalidad para crear, configurar, recuperar, actualizar y eliminar instancias de herramientas externas, gestionando así el **ciclo de vida de las herramientas externas** [Gom09], el cual es común a muchas de las herramientas software, incluyendo las nativas de los VLE. Este ciclo de vida puede combinarse con las características propias de los VLE para la gestión de grupos y actividades, con el objetivo de asociar cada instancia de herramienta externa a cada grupo que participa en una actividad determinada; de esta forma, los estudiantes que pertenecen al mismo grupo pueden colaborar, compartiendo la misma instancia (como generalmente sucede con las herramientas nativas de los VLE). Gracias a esta funcionalidad, se facilita en gran medida la instanciación y puesta en marcha de actividades colaborativas e individuales.

Cada una de las tres capas de la arquitectura GLUE! tiene un rol concreto para apoyar la gestión del ciclo de vida de las herramientas externas desde los VLE. Las peticiones de creación, configuración, recuperación, actualización y eliminación de instancias se inician en la interfaz de usuario del VLE, y por tanto, los adaptadores de VLE deben procesarlas y enviarlas como peticiones uniformes al núcleo de GLUE!. Éste las envía a los correspondientes adaptadores de herramientas, los cuales las reciben y procesan, resultando en invocaciones sobre los proveedores de herramientas, siguiendo los contratos establecidos por éstos.

Cada uno de los elementos de las tres capas de GLUE! puede ser ofrecido de forma independiente, excepto los adaptadores de VLE, que normalmente se despliegan junto con los VLE.

El motivo de esto es que los adaptadores de VLE necesitan ser implementados, en la mayoría de casos, como extensiones de los VLE, embebiendo parcialmente la funcionalidad de dichos adaptadores en la interfaz gráfica de los VLE.

Finalmente, es importante mencionar que la arquitectura distingue tres roles: el **administrador** de la instalación de GLUE! que indica en el GLUE! core las herramientas disponibles y sus correspondientes adaptadores; el **educador** que solicita la creación, configuración, eliminación y actualización de instancias de herramientas; y el **participante** que pide la recuperación de las instancias creadas.

Contratos de integración

GLUE! define un contrato de integración para la comunicación entre el núcleo de GLUE! y los adaptadores de VLE y otro contrato de integración para la comunicación entre el núcleo de GLUE! y los adaptadores de herramientas. Estos contratos se definen con el propósito de reducir el esfuerzo de desarrollo y facilitar la adopción de la arquitectura. En estos contratos se incluyen las restricciones impuestas a los proveedores de VLE y herramientas, las tecnologías que deben implementar los adaptadores de VLE y de herramientas, y la funcionalidad ofrecida.

El contrato de GLUE! para VLE impone **tres restricciones obligatorias sobre los proveedores de VLE**: los VLE deben ser capaces de interpretar contenidos web para embeber fácilmente las instancias de herramientas en la interfaz web de los VLE; los VLE deben ofrecer una interfaz de extensión, para que los adaptadores de VLE puedan comunicarse con los VLE sin modificar su código; los VLE deben entender el concepto de herramienta. Además, hay dos requisitos opcionales: los VLE deberían entender el concepto de grupo, y también el concepto de rol, para beneficiarse en mayor medida de la funcionalidad ofrecida por la arquitectura GLUE!. Los principales VLE cumplen tanto los requisitos obligatorios como los opcionales.

El contrato de GLUE! para herramientas solamente impone **un requisito obligatorio sobre los proveedores de herramientas**: el código que las herramientas externas proporcionan para permitir el acceso a su funcionalidad debe poder distribuirse como un contenido web. Además, se define una restricción opcional: las herramientas deberían ofrecer una interfaz programática que permita la creación de instancias configuradas de forma distinta para diferentes usuarios. Curiosamente, el 69 % de las 100 herramientas más populares para educación (*Top 100 Tools for Learning*)[30] se distribuyen online, cumpliendo así el requisito obligatorio. Además, muchas de estas herramientas cumplen también el requisito opcional. Estos números son mayores si se excluyen los dispositivos hardware de esta lista.

[30] `http://c4lpt.co.uk/recommended/2011.html`. Última visita: junio 2012.

Los contratos de integración de GLUE! definen cuatro requisitos tecnológicos que los adaptadores de VLE y de herramientas deben considerar para poder ser interoperables con el núcleo de GLUE!. Todos estos requisitos están basados en tecnologías populares y poco acopladas:

- Los adaptadores de herramientas deben proporcionar una interfaz **RESTful** [Ric07] para ser invocados por el núcleo de GLUE!, mientras que los adaptadores de VLE deben ser capaces de invocar la interfaz **RESTful** ofrecida por el núcleo de GLUE!.
- Los adaptadores de VLE y de herramientas deben ser capaces de procesar peticiones y respuestas en el formato de datos **Atom** [NWG05a].
- Los adaptadores de herramientas deben proporcionar plantillas de configuración en el formato **XForms** [W3C09a] o, en su defecto, en **HTML5** [W3C12]. Además, los adaptadores de VLE deben ser capaces de procesar las plantillas siguiendo estos formatos en los casos en los que los navegadores web no sean capaces de procesar su contenido directamente.
- Los adaptadores de herramientas deben permitir identificar a las instancias de herramientas externas siguiendo la representación **URL**, incluso para aquellas herramientas que no se distribuyan como aplicaciones web.

Finalmente, los adaptadores de herramientas deben distribuir su funcionalidad en dos tipos de recursos REST: *configuration*, el cual permite la recuperación de plantillas de configuración; e *instance* el cual implementa los cuatro tipos de peticiones relacionadas con la gestión del ciclo de vida de las herramientas externas (creación y configuración, recuperación, actualización y eliminación de instancias). Los adaptadores de VLE pueden enviar peticiones a los recursos expuestos por el núcleo de GLUE!: *instance*, sobre el cual se implementan también las cuatro peticiones antes mencionadas, para ser redireccionadas al correspondiente adaptador de herramienta; y *tool*, el cual permite le recuperación de información sobre las herramientas disponibles, y también la solicitud de plantillas de configuración (solicitud redireccionada a su vez al adaptador de herramientas correspondiente). Más información sobre los contratos de integración puede encontrarse en la versión en inglés de este documento.

Conclusiones

El apoyo que GLUE! proporciona a la gestión del ciclo de vida de las herramientas externas trata de facilitar la instanciación (REQ1) y puesta en marcha (REQ2) eficiente de actividades de aprendizaje colaborativo en las que se usan estas herramientas. La imposición de pocas restricciones populares sobre VLE y herramientas externas trata de facilitar la integración de muchas (REQ4) herramientas existentes, entre ellas las más populares (REQ3), en los principales VLE.

Además, la definición de una propuesta de integración poco acoplada que combina interfaces REST, formato de datos Atom, plantillas de configuración XForms o HTML5 y representación URL, la propuesta de una capa intermedia de software que asume parcialmente la funcionalidad de la integración, y el fomento de una integración muchos a muchos tratan de reducir el esfuerzo de desarrollo (REQ5). Finalmente, por medio del patrón adaptador, los VLE y las herramientas existentes pueden ser adaptadas e integradas sin modificar su código (REQ6).

La arquitectura GLUE! puede ser vista como una propuesta de integración alternativa a aquellos trabajos de integración muy acoplados que requieren un esfuerzo de desarrollo elevado. Sin embargo, GLUE! también puede combinarse con otros trabajos de integración poco acoplados, como Apache Wookie y Basic LTI. El motivo es que estos trabajos utilizan tecnologías similares y la funcionalidad que ofrecen encaja dentro del ciclo de vida de las herramientas externas apoyado por GLUE!. Además, a pesar de que GLUE! es una solución de integración para el contexto de las herramientas software y los VLE, también podría aplicarse sobre otras plataformas utilizadas para al aprendizaje colaborativo, tales como wikis, redes sociales o PLE.

4. GLUE!-RI: Implementación de referencia de la arquitectura

Una vez que la arquitectura GLUE! ha sido completamente especificada, cualquier desarrollador puede proceder a su implementación. Sin embargo, para facilitar este trabajo, y también para poder evaluar GLUE! en escenarios reales, se ha desarrollado una implementación de referencia de GLUE!, denominada GLUE!-RI. GLUE!-RI es el término genérico de una distribución software que incluye una implementación de referencia del núcleo de GLUE!, y varios ejemplos de adaptadores de VLE y de herramientas.

El desarrollo de una implementación de referencia de GLUE! en este punto es altamente recomendable por varios motivos. Primero, permite mostrar que la arquitectura puede ser implementada. Además, GLUE!-RI puede ser probada y evaluada para estudiar si GLUE! supera las limitaciones de los trabajos relacionados y cumple con los requisitos de los principales actores interesados. Igualmente, puede ser utilizada por educadores y estudiantes reales para soportar situaciones de aprendizaje colaborativo, siendo éste el propósito final para la arquitectura. Por último, puede servir para iniciar una colección de adaptadores de VLE y de herramientas que crezca con las contribuciones de los desarrolladores externos. Esto es muy importante, ya que los trabajos de integración genéricos normalmente confían en que se creen comunidades de desarrolladores que compartan sus implementaciones, contribuyendo así a incrementar el número de sistemas integrados.

Es importante destacar que GLUE!-RI es solamente la implementación de referencia y que por tanto otras implementaciones son posibles. De hecho, cualquiera de las tres capas de GLUE!-RI (núcleo de GLUE!, adaptadores de VLE, adaptadores de herramientas) podría ser mejorada para permitir nuevos casos de uso, construidos sobre los ya existentes, así como para integrar nuevos VLE y herramientas, mientras se cumpla con los contratos de integración de GLUE!. El código de GLUE!-RI está disponible en `http://gsic.uva.es/glue` para su mejora y reutilización a la hora de programar nuevos adaptadores.

Visión general de la implementación de referencia

GLUE!-RI se ha desarrollado de forma iterativa e incremental. En el momento de escritura, están disponibles como parte de GLUE!-RI el núcleo de GLUE!, tres adaptadores de VLE y nueve adaptadores de herramientas. Los tres adaptadores de VLE permiten la integración de herramientas externas en Moodle, LAMS y MediaWiki[31]. Los nueve adaptadores de herramientas permiten la integración en VLE de Google Docs[32] (Documents, Spreadsheets and Presentations), MediaWiki, Dabbleboard[33], *widgets* W3C desplegados en servidores Apache Wookie, Doodle[34], Facebook Live Stream[35] (un foro para usuarios de Facebook), Kaltura[36], Noteflight[37] y cualquier URL representando a un contenido web.

GLUE!-RI tiene licencia GPL[38] (*GNU General Public License* - Licencia Pública General de GNU) para usos no comerciales. Por tanto, la implementación de referencia de los elementos disponibles puede ser redistribuida y modificada bajo los términos establecidos por esta licencia. Aquellos interesados en emplear alguno de estos elementos para usos comerciales deben contactar con el Grupo de Sistemas Inteligentes y Cooperativos (GSIC), propietario de GLUE!-RI.

Conclusiones

En este punto, GLUE!-RI es útil para demostrar tres importantes afirmaciones realizadas a lo largo de la tesis sobre GLUE!. La primera es que GLUE! puede integrar herramientas externas existentes en VLE existentes. La segunda es que GLUE! puede integrar herramientas externas en otras plataformas en las que el apoyo a la colaboración es relevante; es el caso de MediaWiki con su correspondiente adaptador de VLE. Finalmente, GLUE! puede ser utilizado como arquitectura *middleware* para interoperar con otras aproximaciones de bajo acoplamiento; aquí el ejemplo es el adaptador de herramienta para los *widgets* W3C desplegados en servidores Apache Wookie.

Finalmente, es importante destacar que esta implementación de referencia es útil para llevar a cabo la evaluación de GLUE!, y ver si se cumplen los requisitos de los principales actores interesados y se superan las limitaciones de trabajos previos. Para ello, a continuación se resume brevemente la metodología empleada, las experiencias realizadas, y los resultados obtenidos como parte de esta evaluación.

[31] `http://mediawiki.org`. Última visita: junio 2012.
[32] `http://docs.google.com`. Última visita: junio 2012.
[33] `http://dabbleboard.com`. Última visita: junio 2012.
[34] `http://doodle.com`. Última visita: junio 2012.
[35] `http://developers.facebook.com/docs/reference/plugins/live-stream`. Última visita: junio 2012.
[36] `http://kaltura.com`. Última visita: junio 2012.
[37] `http://noteflight.com`. Última visita: junio 2012.
[38] `http://www.gnu.org/licenses/gpl.html`. Última visita: junio 2012.

5. Evaluación

La evaluación de la arquitectura GLUE! se ha diseñado para cubrir tanto la dimensión tecnológica como la dimensión educativa presentes en esta investigación. En esta evaluación se ha utilizado el marco CSCL-EREM (*CSCL Evaluand-oriented Responsive Evaluation Model* - Modelo de Evaluación receptivo orientando al evaluando) [Jor09]. Este marco facilita a los investigadores y a los profesionales de la educación la evaluación formal de cursos, recursos, estrategias de aprendizaje y sistemas software en contextos CSCL [Jor09], siendo este último el caso de GLUE!. El CSCL-EREM está centrado en el concepto *evaluand* (evaluando), que representa lo que se quiere evaluar (es decir, el cumplimiento de los requisitos de los principales actores interesados en el contexto de esta tesis), y proporciona una representación gráfica gracias a la cual pueden verse de un vistazo el contexto, los objetivos y los métodos relacionados con aquello que se quiere evaluar.

Utilizando el CSCL-EREM se ha formalizado la evaluación de cuatro experiencias realizadas en cursos universitarios. Se ha elegido este tipo de evaluación basada en experiencias auténticas para mostrar que GLUE! puede ser utilizado de forma regular para llevar a cabo situaciones de aprendizaje colaborativo que requieran la integración de herramientas externas. Además, la utilización de experiencias auténticas [Dew01] con diferentes tipos de actores, tales como usuarios finales o desarrolladores, es una práctica común para la evaluación de sistemas que apoyan la colaboración [Bot08,Iso10]. Sin embargo, el hecho de involucrar a los usuarios de los sistemas en la evaluación conlleva una dificultad añadida, ya que las oportunidades de llevar a cabo experiencias reales son, en general, muy escasas, y su organización muy costosa.

En lo que se refiere a los métodos de investigación, la dimensión educativa se ha evaluado utilizando el método mixto propuesto en [Mar03], de acuerdo con el cual se recogieron y analizaron datos cualitativos y cuantitativos procedentes de estas experiencias. De acuerdo con este método, los datos cuantitativos no están dirigidos a demostrar hipótesis, sino a detectar tendencias que luego son confirmadas o descartadas utilizando datos cualitativos. Por otro lado, las propiedades tecnológicas de la arquitectura se han evaluado a través de un análisis de características (*feature analysis*) [Kit97b] acompañado de evidencias empíricas acerca de la complejidad del software,

obtenidas con métricas como las líneas de código fuente nuevas que deben desarrollarse (SLOC - *source lines of code*) [Alb83], y el tiempo invertido en su desarrollo.

Cumplimiento de los requisitos

Los resultados de evaluación obtenidos a partir de las cuatro experiencias auténticas permiten concluir que GLUE! cumple con los requisitos de los principales actores interesados. A continuación se presentan estos requisitos y de forma breve las evidencias obtenidas de las experiencias que apoyan su cumplimiento. En la versión en inglés se recogen bastantes más evidencias de apoyo a estas conclusiones y se discuten en mucha mayor profundidad.

- *REQ1: Permitir la instanciación de actividades individuales y colaborativas que requieran la integración de herramientas externas con un esfuerzo asumible para los educadores.* Los datos de evaluación permiten afirmar que GLUE! redujo el tiempo de instanciación en más de un 80 % al combinarse con VLE como Moodle o LAMS, facilitando por tanto la instanciación de actividades individuales y colaborativas en las que fueron necesarias herramientas externas. Esto ha sido corroborado por los educadores que participaron en las experiencias a partir de datos cualitativos y cuantitativos procedentes de cuestionarios y entrevistas.

- *REQ2: Permitir la puesta en marcha de actividades colaborativas que requieran la integración de herramientas externas facilitando la colaboración entre los participantes.* Se han obtenido diferentes evidencias cualitativas y cuantitativas mediante cuestionarios y grupos de discusión con los alumnos que apoyan el cumplimiento de este requisito. Concretamente, más del 77 % de los alumnos consideraba que el soporte tecnológico (el cual incluye un VLE, algunas herramientas externas, algunas herramientas internas, y GLUE!) había facilitado bastante o mucho la colaboración. Asimismo, más del 72 % estaba de acuerdo en que ver las contribuciones de sus compañeros a lo largo de las actividades era fácil o muy fácil gracias a este apoyo tecnológico.

- *REQ3: Permitir la integración de VLE y herramientas existentes y populares.* Los educadores seleccionaron cinco herramientas existentes distintas (Google Documents, Google Presentations, Dabbleboard, Doodle y el *widget* You Decide) y tres VLE (Moodle, LAMS y MediaWiki) en las cuatro experiencias que se llevaron a cabo. Es interesante apuntar que Google Documents y Google Presentations están entre las herramientas mejor valoradas por educadores, según la lista de las 100 herramientas preferidas para educación (*Top 100 tools for learning* del C4LPT), siendo Moodle el VLE con una mayor adopción actualmente.

- *REQ4: Permitir la integración de muchas herramientas externas.* Los educadores tuvieron a su disposición 17 herramientas externas durante la instanciación de las experiencias. Este número por sí sólo es mayor que el número de herramientas internas de muchos VLE como Moodle, Blackboard o Claroline. Además, al menos el 69 % de las herramientas listadas entre las 100 mejor valoradas cumplen el requisito obligatorio impuesto por el contrato de GLUE! para herramientas, y por tanto, son potencialmente integrables.

- *REQ5: Demandar un esfuerzo de desarrollo asumible para la integración de nuevos VLE y herramientas.* El esfuerzo de desarrollo necesario para poner en marcha las distintas experiencias disminuye progresivamente, aun cuando se utilizan diferentes VLE. Además, una vez desarrollados los principales adaptadores de VLE, la integración de herramientas externas requiere un esfuerzo bastante bajo (aproximadamente 100 líneas nuevas de código y 6-8 horas de trabajo por herramienta), ofreciendo un rendimiento alto ya que la nueva herramienta queda integrada en varios VLE (en todos aquellos para los que se ha desarrollado el correspondiente adaptador de VLE con anterioridad). El esfuerzo de los adaptadores de VLE es mayor, aunque similar al necesario en otras propuestas de integración de bajo acoplamiento.

- *REQ6: Ser construido sobre los VLE y las herramientas existentes sin modificarlos.* Todos los adaptadores de VLE y de herramientas disponibles se han desarrollado utilizando las interfaces de extensión e integración proporcionadas por los proveedores de VLE y herramienta. Por tanto, estos adaptadores no modifican la implementación existente, y son compatibles con las distribuciones oficiales de los VLE y de las herramientas integradas.

Comparación con otros trabajos de integración de bajo acoplamiento

El análisis de características realizado sobre GLUE!, Apache Wookie [Wil08] y Basic LTI [IMS10b] sirvió para comparar la funcionalidad y características de estas propuestas de integración de bajo acoplamiento. Como consecuencia de ese análisis, puede concluirse que GLUE! ofrece un mayor grado de funcionalidad en comparación con Basic LTI. Además, el grado de funcionalidad ofrecido por GLUE! es similar al que ofrece Apache Wookie. Sin embargo, GLUE! permite la integración de herramientas externas heterogéneas, a diferencia de lo que ocurre con Apache Wookie (el cual requiere que todas ellas cumplan con la especificación W3C [W3C11]).

En cuanto al esfuerzo de desarrollo, cabe destacar que hay diferencias menores en lo que respecta a la implementación de adaptadores para Basic LTI (consumidores y proveedores en la terminología propia de esta aproximación), y adaptadores de VLE y de herramientas para GLUE!, respectivamente. En cambio, Apache Wookie sí que requiere un esfuerzo inicial menor en

comparación con GLUE!. Sin embargo, si los *widgets* van a ser utilizados desde alguno de los VLE compatibles con GLUE!, entonces puede merecer la pena desarrollar un adaptador que permita la comunicación entre el núcleo de GLUE! y el servidor de Apache Wookie. De hecho, esto es lo que sucedió en la implementación de referencia de GLUE!, permitiendo que estos *widgets* puedan ser integrados actualmente en Moodle, LAMS y MediaWiki. Esto refuerza la idea de que GLUE! puede funcionar como una arquitectura *middleware* compatible con otras aproximaciones de bajo acoplamiento, como las citadas Basic LTI y Apache Wookie. Finalmente, también se obtuvieron evidencias del esfuerzo de desarrollo para algunas aproximaciones *ad hoc* con un mismo grado de acoplamiento y funcionalidad que GLUE!, mostrando que una aproximación *ad hoc* es la peor opción en términos de esfuerzo, especialmente si la herramienta va a ser usada desde distintos VLE. En la versión en inglés se proporcionan más detalles sobre esta comparativa.

Conclusiones

La evaluación de la arquitectura GLUE! permite afirmar que se cumplen los objetivos de los principales actores interesados, y también que se superan las limitaciones encontradas en otros trabajos de integración en este contexto. Por tanto, una vez que se ha completado la evaluación satisfactoriamente, puede considerarse que el objetivo global presentado al comienzo de la tesis ha sido alcanzado. A pesar de ello, quedan todavía algunas líneas abiertas que se presentan en el último capítulo recogiendo también las principales conclusiones al trabajo de investigación.

6. Conclusiones y trabajo futuro

Esta tesis ha tratado de dar solución a la limitación encontrada en lo que respecta al número de herramientas ofrecidas por los VLE para el apoyo de situaciones de aprendizaje colaborativo. Esta limitación, mencionada de forma recurrente en la literatura, dificulta la instanciación y puesta en marcha de muchas actividades de aprendizaje, tanto individuales como colaborativas. Tras analizar los problemas de adopción de los primeros trabajos en este campo, los cuales proponían nuevos VLE y herramientas, se ha optado por investigar la integración de herramientas existentes en VLE existentes. Sin embargo, aquellos trabajos que han tratado esta integración también tienen problemas importantes. Uno de ellos afecta al esfuerzo de desarrollo, que siendo necesario para conseguir la comunicación entre VLE y herramientas, es significativamente alto en muchas aproximaciones. Otro problema destacado es el tipo de restricciones impuestas a los proveedores, las cuales impiden que muchos VLE y herramientas populares puedan ser integrados. Finalmente, el tercer problema consiste en que algunos trabajos no tienen en cuenta cómo deben gestionarse las herramientas externas a la hora de instanciar y poner en marcha situaciones de aprendizaje colaborativo, impidiendo así que los profesionales de la educación puedan aprovecharse de las características colaborativas proporcionadas por los VLE.

Las limitaciones encontradas en los trabajos relacionados han sido estudiadas con detalle en el capítulo 2 de este documento, al analizar el problema de la integración, particularizado para un conjunto representativo de VLE y herramientas, siendo éste el primer objetivo parcial de la tesis. El análisis del problema de la integración ha supuesto la identificación de los seis principales requisitos de los actores interesados, y la identificación y discusión de las principales decisiones de diseño y sus alternativas, las cuales deberían ser tenidas en cuenta al proponer nuevas aproximaciones de integración. Estas decisiones de diseño están interconectadas, requiriendo un compromiso entre las alternativas elegidas para intentar cumplir con todos los requisitos de los actores interesados. Las alternativas recomendadas en esta tesis para cumplir con estos requisitos y superar así las limitaciones de los trabajos relacionados son: imponer pocas restricciones populares sobre los proveedores de VLE y herramientas; promover una integración muchos a muchos poco acoplada para reducir el esfuerzo de desarrollo; y ofrecer funcionalidad suficiente para, al menos, facilitar la gestión del ciclo de vida de las herramientas externas. La **identificación de**

los principales requisitos y decisiones de diseño es una contribución original de esta tesis que puede encontrarse en [Ala10a, Ala10c, Ala12a].

Sobre esta base y tratando de alcanzar el segundo objetivo parcial, se propone una arquitectura middleware, llamada GLUE! en el capítulo 3. GLUE!, es una arquitectura de tres capas poco acopladas compuesta por un elemento intermedio y dos conjuntos de adaptadores (de VLE y de herramienta). GLUE! promueve la integración muchos a muchos de VLE y herramientas existentes imponiendo pocos requisitos que los principales proveedores cumplen actualmente. Además, la arquitectura GLUE! está pensada para que terceros puedan integrar nuevos VLE y herramientas desarrollando adaptadores. Con respecto a la funcionalidad ofrecida, GLUE! permite gestionar el ciclo de vida de las herramientas externas, facilitando que se puedan instanciar y poner en marcha actividades individuales y colaborativas desde la interfaz del VLE. **La arquitectura GLUE!**, incluyendo los elementos y sus responsabilidades, los contratos de integración y las restricciones impuestas a VLE y herramientas, ha sido publicada en [Ala12a], siendo la contribución principal de esta tesis.

La arquitectura GLUE! se ha desarrollado como una implementación de referencia denominada GLUE!-RI (véase el capítulo 4), tal y como establecía el tercer objetivo parcial de esta tesis. GLUE!-RI es una distribución software que incluye la implementación de referencia del núcleo de GLUE! y un conjunto de ejemplos de adaptadores de VLE y de herramientas. GLUE!-RI demuestra así que la arquitectura GLUE! no es sólo una propuesta teórica sino que también es un sistema software que puede ser usado por educadores y estudiantes en escenarios educativos reales. El código de GLUE!-RI (disponible en `http://gsic.uva.es/glue`) puede ser descargado e instalado por cualquiera que esté interesado en utilizar GLUE! para la integración de herramientas externas. Además, el código fuente se distribuye con licencia GPL para ser reutilizado a la hora de desarrollar nuevos elementos para la arquitectura GLUE!. **GLUE!-RI** es la tercera contribución de este trabajo, siendo mencionado en primera instancia en [Ala12a], y posteriormente particularizado para los casos de Moodle y LAMS en [Ala11b, Ala12c].

La evaluación de esta tesis tenía como objetivo mostrar que GLUE! cumple con los seis requisitos de los actores interesados (véase el capítulo 5). Esta evaluación se ha apoyado en el marco CSCL-EREM y se han definido para ella cuatro experiencias involucrando a educadores y estudiantes de nivel universitario en distintos contextos. Estas experiencias sirvieron para recoger datos y evidencias que apoyasen la evaluación. Dichos resultados permiten afirmar que GLUE! cumple con los requisitos determinados al inicio de la tesis, lo que se espera fomente su adopción por parte de los diferentes actores. Estos experimentos permitieron observar, por ejemplo, que GLUE! puede facilitar la instanciación y puesta en marcha de actividades colaborativas, reduciendo el tiempo de instanciación en más de un 80 % (al combinarse con Moodle o LAMS). Además, GLUE! puede integrar muchas herramientas (al menos diecisiete actualmente) y no requiere abandonar los VLE a los que educadores y estudiantes están ya acostumbrados. Se es-

pera por tanto que estas ventajas motiven a los profesionales de la educación a adoptar esta arquitectura. En lo que respecta a otros actores, GLUE! trata de estimular las contribuciones de los desarrolladores al reducir el esfuerzo de desarrollo, debido al bajo acoplamiento, a su capa intermedia de software y a su integración muchos a muchos. De forma significativa, estas contribuciones no requieren cambios en el código original de VLE y herramienta, tal y como solicitan los proveedores. **La evaluación de la arquitectura GLUE! y GLUE!-RI** ha sido publicada en [Ala12a, Ala12b], y completa el último objetivo parcial de esta tesis.

Con todo ello, es posible afirmar que se ha alcanzado el objetivo global de esta tesis: **Diseñar, desarrollar y evaluar una arquitectura *middleware* que permita la integración de múltiples herramientas externas existentes en múltiples VLE existentes, requiriendo un esfuerzo de desarrollo asumible para integrar nuevas herramientas y VLE, imponiendo sólo restricciones básicas que la mayoría de proveedores de VLE y herramientas cumplan, y ofreciendo suficiente funcionalidad para facilitar la instanciación y puesta en marcha de situaciones de aprendizaje colaborativo**.

Trabajo futuro

Durante la realización de esta tesis han surgido algunos temas que han sido anotados para futuras revisiones de esta propuesta. Además, algunos otros temas quedaban fuera del ámbito principal de esta investigación, y por ello se marcaron con una prioridad más baja. En este punto se sugieren varias líneas de trabajo que podrían extender y mejorar la investigación presentada en esta tesis, y que incluyen algunos de estos temas. Para entender mejor las líneas de trabajo, éstas se han clasificado en cuatro categorías: integración de nuevos sistemas, mejora de los sistemas integrados, integración con otros sistemas en el ciclo de vida CSCL y programas de transferencia.

Dentro de la **integración de nuevos sistemas** destacan tres líneas de trabajo. La primera de ellas es la integración de nuevas plataformas de aprendizaje como Sharepoint LMS o Sakai. Dentro de esta misma línea también se recoge la migración a nuevas versiones de los VLE ya integrados, y el estudio de la integración de herramientas en otras plataformas distintas a los VLE, como pueden ser los CMS o los PLE. La segunda línea es la integración de nuevas herramientas externas. Sobre esto ya se está trabajando en algunos casos concretos, como por ejemplo la integración del editor de texto colaborativo PiratePad. Sin embargo, dado que el esfuerzo de desarrollar adaptadores de herramientas no es muy elevado, se espera poder establecer prioridades a petición de los propios educadores. Queda pendiente también como parte de esta línea, el desarrollo de un adaptador que permita alcanzar la compatibilidad entre GLUE! y Basic LTI, permitiendo integrar herramientas externas que cumplan con esta especificación de IMS. Finalmente, una tercera línea es la de fomentar el desarrollo de adaptadores por parte de terceros. En este sentido ya se han establecido contactos con varias empresas y universidades para este fin.

Como parte de la **mejora de los sistemas integrados** se incluyen algunas líneas de trabajo futuro al margen del objetivo principal de esta tesis que necesitan una mayor reflexión, y también algunas tareas de implementación concretas de baja prioridad, que por falta de tiempo no han podido completarse. Un ejemplo de las primeras es la revisión de las propuestas de autorización de nivel de usuario para la gestión de herramientas externas. Actualmente, se plantean dos soluciones de compromiso para este problema de seguridad que se han implementado, aunque ambas tienen limitaciones en entornos con muchos usuarios o poco controlados. Además, dentro de las tareas de implementación pendientes destacan, entre otras: la implementación de las propuestas diseñadas para los otros problemas de seguridad detectados (incluyendo diferentes niveles de autorización y también la privacidad en las comunicaciones); el desarrollo de un cliente de administración para el registro interno de herramientas; la adopción de HTML5 como formato para las plantillas de configuración en los adaptadores ya desarrollados; el apoyo a la creación de copias de seguridad de cursos que incluyan herramientas externas en Moodle (como parte de la lógica del adaptador de este VLE); y la funcionalidad para reutilizar instancias de herramientas externas en distintas actividades de una misma lección en LAMS.

Otro conjunto de líneas de trabajo futuro se engloban como parte de la **integración de la arquitectura GLUE! con otros sistemas dentro del ciclo de vida CSCL**. En este sentido, una de las investigaciones en curso promueve la utilización de una arquitectura denominada GLUE!-PS (*GLUE!-Pedagogical Scripting* - GLUE!-Guiado Pedagógico) [Mun12b,Pri11] para el despliegue de diseños de aprendizaje generados con múltiples herramientas de autoría y lenguajes de modelado en múltiples VLE. Estos diseños pueden particularizarse también utilizando herramientas externas integradas mediante la arquitectura GLUE!. La alianza GLUE! - GLUE!-PS ya ha sido probada en experiencias con educadores reales, los cuales destacaron los beneficios de esta aproximación conjunta [Ala12d]. Otra línea de trabajo que actualmente se está desarrollando es la posibilidad de que los profesores filtren las herramientas externas en las que están interesados mediante búsquedas semánticas; esto es especialmente útil a medida que el número de herramientas integradas crece. También como parte de esta línea se ha propuesto una infraestructura basada en datos enlazados (*linked-data*) [Biz09], denominada SEEK-AT-WD (*Support for Educational External Knowledge About Tools in the Web of Data* - Apoyo al Conocimiento Educativo Externo sobre Herramientas en la Web de Datos) [Rui12a,Rui12b] que puede ser útil para reducir la carga de administración dedicada a poblar y mantener el registro interno de herramientas de GLUE!. También se está trabajando en una extensión para GLUE! denominada GLUE!-CAS (*GLUE!-Collaboration Analysis Support* - GLUE!-Apoyo al Análisis Colaborativo) [Rod11,Rod12] orientada a obtener información de distintas fuentes (VLE, GLUE!, herramientas externas) sobre análisis de interacciones [Sol05], necesaria para facilitar la evaluación del trabajo de los alumnos. Por último, otra línea de trabajo pendiente es el uso de geolocalización en las de herramientas integradas mediante GLUE! [Mun12a]. Esto permitiría asignar coordenadas (latitud y longitud) a las diferentes instancias, de tal forma que los estu-

diantes, utilizando dispositivos móviles, deberían posicionarse cerca de dichas coordenadas para poder visualizar el contenido de las instancias.

Finalmente, es importante destacar que éste es un trabajo de investigación aplicado que se espera sirva para mejorar las prácticas educativas. Por ello, es muy importante definir **programas de transferencia** con instituciones dedicadas a la enseñanza. En este sentido el primer paso dado ha sido el desarrollo de una implementación de los elementos de la arquitectura que cualquiera puede descargar de `http://gsic.uva.es/glue` e instalar. Además, se han creado manuales de usuario y vídeos explicativos (accesibles también a partir de ese enlace) con las novedades que introduce la arquitectura en el funcionamiento habitual de los VLE. Finalmente, se han llevado a cabo negociaciones para la adopción de esta propuesta en varias instituciones como la Universidad de Valladolid, el Instituto de Oftalmobiología Aplicada (IOBA) asociado a esta Universidad y la Universidad de Mondragón.