The Opportunity of Grid Services for CSCL-Application Development

L.M. Vaquero-González, D. Hernández-Leo, F. Simmross-Wattenberg, M.L. Bote-Lorenzo, J.I. Asensio-Pérez, Y.A. Dimitriadis, E. Gómez-Sánchez, G. Vega-Gorgojo
University of Valladolid, Spain
{lvaqgon@ulises., davher@, fedesim@lpi., migbot@, juaase@, yannis@, edugom@, guiveg@}tel.uva.es

Abstract

The choice of the most suitable middleware technology for a specific problem or domain is sometimes erroneously based on current trends instead of on a thorough comparison of the features offered by the different options. This could be the case of grid services technology, which is claimed to be a technological advance, but whose characteristics are not clearly contrasted with other middleware technologies. The aim of this paper is to analyze whether grid services technology truly present some required properties for CSCL (Computers Supported Collaborative Learning) application development in comparison with other service-oriented middleware technologies, as well as with other types of middleware paradigms (object-oriented or component-oriented). To this end, we present relevant requirements of CSCL applications and theoretical discussion about how well they are satisfied by the aforementioned paradigms and technologies. Finally, we introduce a case study in order to illustrate our conclusions.

1. Introduction

Computer-Supported Collaborative Learning (CSCL) is a research paradigm within the field of educational software that underlines the key role that social interactions play in the process of learning [8,20]. CSCL applications must support several requirements in order to be effective in terms not only of collaborative learning theories and practices but of economic factors as well. These requirements range from typical features as co-ordination, communication and collaboration [11] to the reuse of software pieces [10,31] and the flexibility inherent to any learning process [7,22].

The development of such applications can be simplified by leveraging middleware, in that it resolves the problem of heterogeneity, and facilitates communication and co-ordination of distributed elements [3,13]. Hence, in order to build distributed systems that meet these requirements, software engineers must know what middleware is available and which one is best suited to the problems at hand [37]. However, the proliferation and continuous evolution of middleware technologies have highlighted this selection problem [4,33]. There are not only different middleware paradigms such as those oriented to objects [13], components [34] or services [29], but also different specifications [26,27,32] for each of those paradigms.

Significantly, a new technology has recently appeared: grid services (GS) [2]. This technology belongs to the service-oriented middleware (other example is web services technology (WS) [5]) paradigm and enables the deployment of grid infrastructures [1] in which multiple organizations can share heterogeneous resources. In this infrastructures resources ranging from data, files, or programs to sensors, scientific instruments, display devices, computers, and supercomputers are offered by providers in the form of grid services. Applications in a grid context can use shared resources for any purpose. Typical examples include leveraging the computational power provided by a large number of computers or accessing specific hardware/software [15].

Grid technology thus could be a good candidate for developing CSCL applications, and specially those that may require supercomputational capabilities such as in [19,30] or access to specific hardware resources such as in [6,23]. Nevertheless, the selection of this technology should not be based more on market trends but on the results of a thorough comparisons of alternatives.

Within this context, the main goal of the paper is to analyze if GS technology actually provides the

capabilities required by CSCL applications, and compare them with those offered by other middleware technologies. Three partial objectives can be derived from this global aim:

- To identify key requirements of CSCL-application development. This involves designer, software engineer, programmer, administrator, and every technical role related to applications' lifecycle from design to maintenance.
- To state whether service-oriented middleware satisfies these requirements and make a comparison with object and component-oriented middleware.
- To analyze whether GS technologies satisfies the identified requirements in comparison with web service technologies.

The structure of this paper is as follows. Section 2 highlights CSCL requirements and a brief background of middleware technologies from an evolutionary point of view. Section 3 is devoted to a comparative analysis. First, service-oriented middleware is compared to object and component-oriented middleware paradigms. Then, a comparison between GS and WS follows. In section 4, an overview of a grid service-based CSCL system being developed by our research group is shown, which is employed in order to illustrate some of our assertions. Conclusions and future work are presented in section 5.

2. Formulation of the Problem

As mentioned above, CSCL focuses on the use of technology as a means of supporting collaborative methods of learning [20]. This section points out CSCL requirements and presents a brief outline of well-known middleware paradigms and technologies.

2.1 CSCL Requirements

Different pedagogical approaches and strategies require a high degree of flexibility in CSCL applications, due to the existing dynamism in educational environments, namely teachers' preferences, students' capabilities and evolution (even in the same course), etc. [9]. In this way, CSCL applications may require dynamic adaptation to situations that might arise at run time in a learning scenario. For instance, the next task to be done may depend on the result of the previous tasks, or planned educational objectives may be changed during the application lifetime (depending on students' abilities, etc.) [31].

Moreover, CSCL applications are often so specific that they cannot be reused nor integrated within a single framework even though they share many common features. Since CSCL development is a complex process, this development effort is only justified when we have the chance to reuse previous works into newer ones. It would be highly desirable to have standard protocols and interfaces in order to hasten application variety through reusability [10] of software pieces.

Educators are often constrained by CSCL applications when planning (or even performing) their desired learning scenarios. Fulfilling actual CSCL pedagogical needs may require run time modification of the educational script [7] according to a set of predefined paths specified by the teacher. Tailorability [25] is the capability of a system to find suitable tools and to integrate them in collaborative environments as specified by educators in the scripts. In this sense, the integration of these tools is accomplished by educators or students, and not by technical staff. These final users are not supposed to have high technical knowledge. So, technical issues should somehow be hidden, thus allowing the teacher to concentrate on a proper educational design. This is known as the technification problem [25].

Modern CSCL applications require more sophisticated gadgets enabling the usage of astronomy telescopes, physics simulators, Virtual Reality collaborative education, etc. The cost of such devices (like powerful telescopes, supercomputers, and many others) is often unaffordable for small institutions. Besides, a single institution might have a huge number of highly heterogeneous software and hardware resources to satisfy every educational situation that might appear [6,30].

In addition, scientific collaboration and education require user interaction through a shared environment (common resources), and perhaps across different administrative domains (cross-organizational).

On the other hand, typical collaborative applications include some features: co-ordination, communication, and collaboration [11]. Besides, for two people to collaborate, it is necessary to be aware of what other people within the same group are doing that could enhance collaboration effectiveness (awareness). Awareness and co-ordination (of activities, shared objects, or software) imply a one-to-many communication model in order to be able to take a co-ordination decision, to notify that a new activity has just been started by a user, etc.

2.2 Middleware technologies background

Some authors [13,17] consider Sun's Remote Procedure Calls (RPCs) and Message Oriented Middleware (MoM) as the first middleware platforms, developed in the early 80s. In this paper we are only considering more advanced middleware paradigms, since those early technologies provide just communication facilities for programmers. Figure 1 summarizes the chronological evolution of middleware solutions we will focus on.

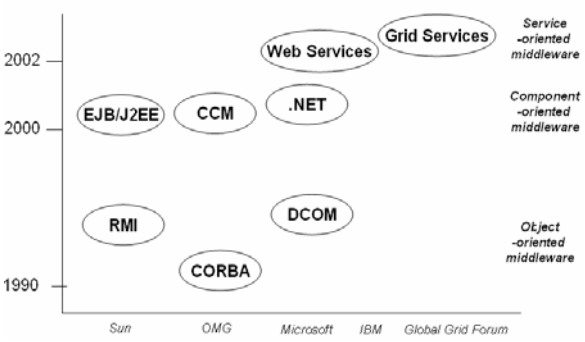

Figure 1: Evolution of studied middleware paradigms and related technologies

More recent middleware technologies focus on the object and component paradigm. Object Middleware (CORBA; RMI, DCOM) introduces the advantages from Object-Oriented programming in distributed applications. Meanwhile, components do not have external dependencies, but are self-contained, deployable pieces of software. Sun started its own approach, and developed EJB (Enterprise Java Beans) [32] to support distributed component-oriented programming in Java. Microsoft and OMG have technologies to offer too, respectively .NET [24] and CCM (CORBA Component Model) [27].

Nowadays, WS middleware has entered the stage, trying to eliminate some of the previous disadvantages of component-oriented middleware. WS are very flexible since they can be supported by a range of lower-level protocols. They are generally based on SOAP [40], a protocol for exchanging information structured as XML documents, typically through HTTP connections. Besides, WS involve interface definition specifications (WSDL [5]) and service publishing/discovery mechanisms.

Finally, GS middleware has recently emerged from the joint of WS and grid computing [15]. They are based on WS technologies, but provide extensions to better support grid computing, such as state management and notification exchange. Thus, the underlying protocols and technologies are the same as in the case of WS: SOAP for information exchanging and XML to provide data structuring (among other possibilities).

In the literature there are some CSCL applications proposals based on some of these standard middleware technologies ([10] is an example). However, the specific characteristics required by CSCL makes some authors propose their own proprietary middleware [21,28] in order to ease the development of collaborative applications. ANTS [16], for instance, is built on top of the JavaBeans component model and provides additional services like a distributed event system and shared data structures.

3. Comparative Analysis

Through this section, we present a comparison between different middleware paradigms and technologies introduced above to determine their suitability to the requirements stated in section 2.1.

Table 1 maps some middleware characteristics, which would be useful (marked with "Yes") or indifferent (-) in order to assist developers to satisfy these application requirements.

In the next section we show how different paradigms help to satisfy CSCL application requirements, offering these middleware characteristics.

Table 1: Useful Middleware Characteristics to Fulfill Application Requirements

Middleware Charac-teristics	***CSCL Requirements***					
	Flexibility	Tailorability	Technification	Reusability	Cross-Org	Coordination/ Awareness
Loosely-coupled	**Yes**	**Yes**	-	**Yes**	-	-
Highly abstract	-	-	**Yes**	-	-	-
Coarse-grained	-	-	**Yes**	-	-	-
Oneto many comm..	-	-	-	-	-	**Yes**
Self-contained	**Yes**	**Yes**	-	**Yes**	-	-
Standard Tech.	-	-	-	**Yes**	**Yes**	-

3.1. Service vs. Component and Object-Oriented Middleware

Service-oriented middleware enable runtime "assembly" of different elements, using standard loosely coupled technologies [5,29]. This way, applications result from the integration of one or many different services, which can be dynamically assembled, as needed, to satisfy concrete application requirements. This capability exceeds previous

middleware approaches (object and component-oriented middleware) in which we would need to recompile or redeploy [13] respectively (stopping the application), each time a new object or component must be added to our application. In this sense, services enhance the flexibility needed in CSCL applications because it is possible for users without technical knowledge (i.e. students or educators) to integrate the tools required in a specific learning situation. In this point it is important to remember that this integration model is not provided by Peer-to-peer technology (P2P).

A CSCL system must allow teachers to perform these dynamic changes according to several factors (see section 2.1), i.e., the system must be tailorable at run or design time. This may be done using services. Services can be dynamically added to a set in order to create an application. The simplest approach could be an application on the teacher's side including command facilities. This enables them to order clients on the side of students to download a new service client from a URL and start working with that service. More sophisticated functionality could be achieved when a flow language [18] rules the set of used services.

Furthermore, services usually employ (though they do not need to) a high level of abstraction, closer (than objects or components) to non-technical people's mental schemas [31]. This way, an application is more easily integrated by non-technical staff, since they can understand what services do much better than what objects or components do, reducing the technification problem. Besides, this higher granularity makes it easier to "assembly" software pieces in order to integrate the whole application. Obviously, one could build highly abstract objects or components but, anyway, teachers would require technical assistance to customize their application.

Services promote software reusability, since the same service could be involved in many different applications at the same time. Let's consider a group management service employed simultaneously by a medical application for collaborative telediagnosis and a collaborative flight simulator. The service would manage group information regardless of the application that makes the request, and without reprogramming.

While services foster run-time software reusability, software components offer deployment-time and objects compilation-time reusability. These capabilities are really useful from a programmer's point of view, because they can reuse an object or component instance, or a service, as many times as needed. Reusability in services surpasses reusability in object or component oriented middleware, because an application can integrate services belonging to external entities (using standards like HTTP, or XML, for instance), enabling cross-organizational assembly of services to build an application.

Summarizing, standardization, run-time flexibility and coarse granularity employed by service middleware results in better tailorability for users without technical knowledge, and, consequently, a smaller technification problem. From application programmers' point of view, service middleware allows them to use third party services, so they do not have to deal with service development, nor service deployment, nor service maintainability. Service programmers would be in charge of these technical tasks, saving time by using services provided by other organization. Consequently, the task of a service-based application programmer is to select and put together a suitable set of services. Non-technical staff could easily accomplish this task (using coarse-grained services). This makes a difference with other middleware technologies, where technical staff is in charge of almost every development stage [13].

3.2 Grid Services vs. Web Services

Services offer some advantages over previous middleware paradigms for CSCL application development. Even though GS and WS share the same middleware paradigm (service-oriented), both include many subtle differences that could be important for a concrete application domain, like CSCL.

WS are persistent (non-transient) services, i.e., there is a single service of each type in each machine waiting for all-client requests until an authorized administrator finishes them (manages their lifecycle). Collaborative applications often require the support of personal (but simultaneous) activities to achieve a common goal [11]. Not only is this need for individual services important, but also it arises a concurrency problem when different clients simultaneously use the same service. GS try to solve these problems including the concept of factories. Factories permit to create new service instances, instead of using a single service. Factories imply a higher software reusability and easier concurrency management.

In spite of cross-organizational interoperability, WS lack standardization for advanced features to use heterogeneous hardware resources as those that CSCL applications may require [36]. Again, GS fulfill this lack, offering common interfaces to hide heterogeneity [14]. GS ease development of complex applications using heterogeneous hardware resources, because

these interfaces allow a standard access and control mechanism to such devices. Among these heterogeneous resources, supercomputing capabilities can also be found.

Communications from one to many are essential in most synchronous collaborative applications, since they would be helpful to build awareness and co-ordination mechanisms. Unfortunately, SOAP message orientation [40] does not offer such capabilities. In this sense, GS outperform WS functionality, since notifications are already included in the standard specification [14].

Both, web and GS rely on WS standards to support service co-ordination features (BPEL4WS [18] is an example). This co-ordination could be considered equal to activity co-ordination [12], as long as services keep an appropriate level of abstraction.

WS are generally SOAP-based [5], so they are stateless entities [40]. In most collaborative applications there should be a mechanism to keep track of the information representing the actual state of collaboration activities. Besides, awareness facilities require some mechanism to store persistent information to be delivered to groups or users. This enables to keep them aware of what is going on that is interesting to achieve a common goal. GS offer such mechanisms to keep state information [35], easing the development of awareness support for collaborative applications.

Underlying technologies and standards supporting WS and GS, are almost the same (see Table 2).

Table 2: Service-Oriented Technologies Comparison

	WEB SERVICES	***GRID SERVICES***
Information Access	Programmatically	Programmatically
Technology	Web	Web
Platform Independence	High	High
Localization	URI	URI
Communication	Overloaded	Overloaded
State	Stateless	**Stateful** / Stateles
Heterogeneous Hardware Resources	No	**Yes**
Persistency	Persistent (non-transcient)	**Persistent or not**

Since standard web technologies are the pillars, they offer quite similar mechanisms for communication, localization, and similar independence and interoperability capabilities. However, there are small differences (state, persistency and heterogeneity) that make, as we have explained above, GS slightly more appropriate than WS for CSCL development. For instance, GS provide highly dynamic information mechanisms (based on state information), easing the development of some required CSCL features like awareness, and co-ordination.

3.3 Discussion

Object-oriented and component-oriented middleware do not fulfill some important CSCL requirements. They lack run-time flexibility and high granularity (high abstraction level), which results in worse tailorability for users without technical knowledge and, consequently, in a bigger technification problem. On the other hand, not only are object and component middleware mature, but also they are still evolving to include some interesting features for CSCL. For instance state, notifications, or factories in GS are not a new concept, they were used in some object and component middleware as well [26,32]. While CORBA is evolving too, it already includes cross-organizational facilities to use remote objects [38].

Subtle differences between WS and GS can save CSCL application programmers' effort. Both rely on the same standard technologies and there are recent implementations [36] of Open Grid Service Architecture (OGSA) aiming to fit to WS guidelines, and new WS extensions supporting stateful services and notifications [39]. These trends suggest that both approaches are likely to converge, in mid-term future, into a single specification. Besides, since the cradle of GS are high throughput, parallel or data-intensive computation, current state of the art is still focused on computation-demanding applications, making it difficult to develop GS-based production applications. GS de facto standard, Globus Toolkit 3 (GT3) [35], is used for testing only and it is going to be replaced soon [36]. Even though GS paradigm seems more accurate for CSCL development, WS convergence, maturity, documentation and GS current state-of-the-art might make WS more attractive.

In spite of claimed advantages, middleware includes an extra overload to applications. Data and control require additional layers of processing. This is the price to be paid for higher abstraction and programming simplicity. Moreover, middleware may not be suitable enough for highly customized solutions. Middleware vendors try to fulfill as many application requirements as possible, extending their products with new capabilities, and promising effortless development to programmers. Furthermore, specific middleware is built over general-purpose middleware, adding new

dependencies, interoperability problems, and a muddle of products that make it difficult to select the most appropriate one. Consequently, many of them seem to converge, looking so similar, but keeping soft differences which contribute to increase confusion. This makes it difficult to choose a platform, because it usually depends on the application domain, on the application itself, and on developers' preferences. So, technical, and non-technical issues should be exhaustively taken into account.

4. Gridcole Testbed

Grid Collaborative Learning Environment (Gridcole) [2] is GS-based system that enables easy integration of CSCL applications. More specifically, Gridcole provides a search facility through a simple Web Portal that enables teachers with no technical skills to find and integrate the required GS-based tools in customized application. This customization is finally expressed in terms of an educational script. This script is interpreted by a control unit (a GS again), taking care of what activities are done by each user, and when. This unit relies on underlying mechanisms (state and notifications) provided by GT3 [35].

Gridcole is currently being used as testbed to validate the claimed advantages in a real environment. Gridcole enables teachers to select, integrate, launch, stop and modify the activities they want to employ in their courses without the intervention of technical personnel. As long as services are abstract enough to be understood as a single activity by teachers, Gridcole eases the technification problem, enabling better tailorability by using highly granular services. Moreover, Gridcole enables the use of resources from other organizations, since services are provided by third parties using GS technologies. Those services are loosely-coupled entities which can be assembled or disassembled at run time, resulting in better flexibility.

So, Gridcole takes the advantages of GS middleware, bringing an environment for application integration that enhances some limitations within CSCL domain. Gridcole has not been completely developed yet, so more thorough tests should be performed in order to achieve stronger practical conclusions.

5. Conclusions

The selection of a middleware technology for application development is not an easy task. This selection becomes more difficult due to the hype surrounding most middleware technologies.

Through this paper we have shown domain-specific advantages of service-oriented paradigm over other middleware paradigms (object-oriented and component-oriented middleware). Service-oriented middleware has proven to be more suitable to satisfy requirements of CSCL application development. However, service-based systems still require assistance tools for non-technicians. Within services paradigm, we have compared the advantages of GS over WS in order to ease the work of technical staff, while satisfying application requirements. Furthermore, mature GS technologies are still oriented to supercomputation. GT3 is considered as a promising experiment, but the real GS-OGSA (WS compatible) implementation, WSRF, is expected to appear in mid 2005. Its maturity will take a bit longer.

None of these technologies and paradigms may be considered to be the "best solution" for CSCL application development since such a decision must always be made in regards of the very specific problem considered. However, the comparisons carried out in this paper provide helpful hints in this sense.

Furthermore, these middleware paradigms show common evolutionary trends, expanding their potential application domains. This may lead to a convergence in their capabilities, not ensuring, however, their compatibility. In addition, each one of these paradigms includes a wide range of technologies, making it even more difficult such a selection.

We have shown a real environment, Gridcole, which benefits from the discussed advantages of GS middleware for CSCL. Furthermore, Gridcole enables CSCL requiring supercomputation capabilities to get benefited from external supercomputational resources. Besides, cross-organizational interactions are highly static in Gridcole, because third party providers do not exist yet. Moreover, Grids should be mainly employed to hide legacy resources, and to offer them in a standard manner. Gridcole builds everything up from scratch, so this seems to fit better in a WS environment. Gridcole relies on GS technology to provide the advantages of service paradigm for applications and developers (using few GS extensions to WS).

Future work includes evaluation in a real educational environment, in order to verify some claimed advantages (especially technification for teachers, and reusability of different services in different educational scenarios). We are also considering whether it would be interesting to fix our implementation to GT3 until the arrival of WSRF, or whether it is worth it to transitorily migrate to WS.

Acknowlegments

This work has been partially funded by European Commission Project EAC/61/03/GR009 and Spanish Ministry of Science and Technology project TIC2002-04258-C03-02. The authors would also like to thank the rest of "Intelligent & Cooperative Systems Research Group" at the University of Valladolid for their support and ideas.

References

[1] Berman, F., Fox, G., and Hey, A. *Grid Computing: Making the Global Infrastructure a Reality*, Chichester, United Kingdom: John Wiley & Sons, 2003.

[2] Bote-Lorenzo, M. L., Vaquero-González, L. M., Vega-Gorgojo, G., Dimitriadis, Y., Asensio-Pérez, J. I., Gómez-Sánchez, E., and Hernández-Leo, D., "A Tailorable Collaborative Learning System that Combines OGSA Grid Services and IMS-LD Scripting", *Proceedings of the 10th International Workshop on Groupware, CRIWG 2004*, San Carlos, Costa Rica, 2004, pp. 305-321.

[3] Brown, A. W., "Mastering the Middleware Muddle", *IEEE Software*, vol. 16, no. 4, July 1999, pp. 18-21.

[4] Charles, J., "Middleware Moves to the Forefront", *Computer*, vol. 32, no. 5, 1999, pp. 17-19.

[5] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S., "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI", *IEEE Internet Computing*, vol. 6, no. 2, 2002, pp. 86-93.

[6] Despres, C. and George, S., "Computer-Supported Distance Learning: An Example in Educational Robotics", *Proceedings of the 9th International PEG Conference*, Exeter, UK, 1999, pp. 344-353.

[7] Dillenbourg, P., "Over-Scripting CSCL: The Risks of Blending Collaborative Learning with Instructional Design", Paper presented at *Three Worlds of CSCL. Can We Support CSCL*, P. Kirschner, Inaugural Address, Open Universiteit Nederland, 2002.

[8] Dillenbourg, P., *Collaborative Learning: Cognitive and Computational Approaches*, Oxford, UK: Elsevier Science, 1999.

[9] Dimitriadis, Y., Asensio-Pérez, J. I., Gómez-Sánchez, E., Martínez-Monés, A., Bote-Lorenzo, M. L., Vega-Gorgojo, G., Vaquero-González, L. M., "Middleware for CSCL: Software Components Framework and Grid Technology Support" (in Spanish), *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, vol. 8, no. 23, 2004, pp. 21-31.

[10] Dimitriadis, Y., Asensio-Pérez, J. I., Martínez-Monés, A., Osuna-Gómez, C., "Component-Based Software Engineering and CSCL: Component Identification and Dimensioning", *Upgrade (digital journal of European Professional Informatics Societies), special issue on e-learning*, vol. 4, no. 5, 2003, pp. 21-28.

[11] Ellis, C. A., Gibbs, G. L., and Rein, G. L., "Groupware: Some Issues and Experiences", *Communications of the ACM*, vol. 43, no. 1, 1991, pp. 39-58.

[12] Ellis, C. A. and Wainer, J., "A Conceptual Model of Groupware", *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work, CSCW 1994*, 1994, pp. 79-88.

[13] Emmerich, W., "Software Engineering and Middleware", *The Future of Software Engineering. ACM Press*, A. Finkelstein (Ed.), 2000, pp. 117-129.

[14] Foster, I., Kesselman, C., Nick, J. M., and Truecke, S., "The Physiology of the Grid", *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and A. Hey (Ed.), Chinchester, UK: John Wiley & Sons, 2003, pp. 217-249.

[15] Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S., "Grid services for distributed system integration", *Computer*, vol. 35, no. 6, 2002, pp. 37-46.

[16] García, P. and Gómez-Skarmeta, A., "ANTS Framework for Cooperative Work Environment", *IEEE Computer*, vol. 36, no. 3, 2003, pp. 56-62.

[17] Geihs, K., "Middleware Challenges Ahead", *Computer*, vol. 34, no. 6, 2001, pp. 24-31.

[18] BPEL4WS Business Processing Execution Language for Web Services, http://www-106.ibm.com/developerworks/library/ws-bpel/, last visit 2004.

[19] Jensen, N., Seipel, S., Nejdl, W., and Olbrich, S., "COVASE: Collaborative visualization for constructivist learning", *Proceedings of the Conference on Computer Support for Collaborative Learning, CSCL 2003*, Bergen, Norway, 2003, pp. 249-256.

[20] Koschmann, T., Paradigm shift and instructional technology. In: *CSCL: Theory and Practice of an emerging paradigm*, ed. Koshmann, T. Lawrence Erlbaum, 1996, pp. 1-23.

[21] Li, D. and Muntz, R., "COCA: Collaborative Objects Coordination Architecture", *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work (CSCW 1998)*, Seetle, Washington, United States, 1998.

[22] Lin, J., Sadiq, W., and Orlowska, M. E., "Using Workflow Technology to Manage Flexible e-Learning Services", *Educational Technology & Society*, vol. 5, no. 4, 2002, pp. 116-123.

[23] Martins-Ferreira, J. M., Alves, G. R. C., Costa, R., and Hine, N., "Collaborative learning in a web-accesible workbench", *Proceedings of the 8th International Workshop on Groupware, CRIWG 2002*, La Serena, Chile, 2002, pp. 25-34.

[24] .Net, http://www.microsoft.com/net/, last visit 2004.

[25] Morch, A., "Three Levels of End-User Tailoring: Customization, Integration and Extension", *Proceedings of the 3rd Decennial Aarhus Conference*, Aarhus, DK, 1995, pp. 157-166.

[26] CORBA: Common Object Request Broker Architecture, http://www.corba.org, last visit 2004.
[27] CORBA Component Model v. 3.0, http://www.omg .org/technology/documents/formal/components.htm, last visit 2004.
[28] Orozco, P., Asensio, J. I., García, P., Dimitriadis, Y., and Pairot, C., "A Decoupled Architecture for Action-Oriented Coordination and Awareness Management in CSCL/W Frameworks", *Proceedings of the 10th International Workshop on Groupware, CRIWG 2004*, San Carlos, Costa Rica, 2004, pp. 246-261.
[29] Papazoglou, M. P. and Georgakopoulos, D., "Service-Oriented Computing", *Communications of the ACM*, vol. 46, no. 10, 2003, pp. 25-28.
[30] Ramamurthy, M. K., Wilhelmson, R. B., Pea, R. D., Louis M., and Edelson, D. C., "CoVis: A National Science Education Collaboratory", *Proceedings of the American Meteorological Society 4th Conference on Education*, Dallas, TX, USA, 1995.
[31] Roschelle, J., DiGiano, C., Koutlis, M., Repenning, A., Phillips, J., Jackiw, N., and Suthers, D., "Developing Educational Software Components", *Computer*, 1999, pp. 50-58.
[32] EJB: Enterprise JavaBeans Technology, http://java.sun.com/products/ejb/, last visit 2004.
[33] Java Message Service (JMS), http://java.sun. com/products/jms/, last visit 2004.
[34] Szyperski, C., "Component Technology - What, Where and How?", *Proceedings of the 25th International Conference on Software Engineering, ICSE 2003*, Portland, Oregon, USA, 2003.
[35] The Globus Project, www.globus.org, last visit 2004.
[36] WSRF: The WS-Resource Framework, http://www.globus.org/wsrf/, last visit 2004.
[37] Thompson, J., "Avoiding a Middleware Muddle", *IEEE Software*, vol. 14, no. 6, 1997, pp. 92-98.
[38] Vinoski, S., "New Features for CORBA 3.0", *Communications of the ACM*, vol. 41, no. 10, 1998, pp. 45-52.
[39] Vinoski, S., "Web Services Notifications", *Column from IEEE Internet Computing*, vol. 8, no. 3, 2004, pp. 90-93.
[40] SOAP v. 1.2, http://www.w3.org/TR/soap12-part1, last visit 2004.