Gerosa, M.A. & Fuks, H. “A Component Based Workbench for Groupware Prototyping”. Proceedings of the 1st Workshop on Software Reuse Efforts, 2nd Rise Summer School – RISS 2008, October, 27-29, 2008.
Available at http://www.groupwareworkbench.org.br

A Component Based Workbench for Groupware Prototyping

Marco Aurélio Gerosa[1] & Hugo Fuks[2]

[1] Computer Science Department – University of São Paulo (USP)
R. do Matão, 1010 - Cidade Universitária
05508-090 – São Paulo – SP – Brasil

[2] Informatics Department – Catholic University of Rio de Janeiro (PUC-Rio)
R. Marquês de São Vicente, 225, Rio de Janeiro, RJ, 22453-900, Brasil

{gerosa@ime.usp.br, hugo@inf.puc-rio.br }

Abstract. *Several software development problems are especially relevant for groupware development. Problems like synchronism, concurrence, sharing and distribution are critical; the area of CSCW (Computer Supported Cooperative Work) is highly interdisciplinary; and business process that define the workgroup dynamics are difficult to model and to support. The area normally requires qualified programmers, which spend most of their time solving low level technical issues. The developed code becomes highly coupled and difficult to evolve. Other areas advanced considerably when programmers were able to experiment and rapid prototype. For example, IDEs that offer interface widgets allow programmer to think in terms of user interaction instead of window construction technical details. In this paper, component based technology is being used in groupware development to try to overcome its intrinsic difficulties and to promote creativity in new products generation. A two-level component workbench organized according to a collaboration model is presented.*

1. Introduction

Engelbart [1968] pointed out the relevance of applications for office automation, hypertext and groups. Desktop applications usage increased because the advances on graphical user interfaces and widget toolkits. These advances made it possible for average programmers to build applications by dragging and configuring components. The coming of HTML, with its simplicity and tolerance to mistakes, and of WYSIWYG editors made it possible to teach hypertext development even in elementary schools. On the other hand, groupware development still requires qualified programmers trained to deal with protocols, connections, resource sharing, distribution, rendering, session management, etc. This limits the number of developers active in the area and misplaces the creativity and efforts of these developers, taking their attention from the creation of solutions to the solving of low-level technical problems [Greenberg, 2007]. The iterative refinement and prototyping of new ideas are made difficult, and the code becomes highly coupled and dependant on the prevailing technology. The developed systems end up being little adherent to the real interaction needs, because it requires a lot of experimentation, investigation and prototyping to discover them [Tse & Greenberg, 2004].

This scenario illustrates the need to better support groupware developers, in order to facilitate systems assembly, prototyping and experimentation. Developers need to build extendable systems, in a way more suited to accompany the evolution of collaboration support and the characteristics of tasks, groups and technologies involved. Non-specialized developers should be able to adapt and reconfigure the solution for their specific needs—a desirable aim, given that there is no way of foreseeing all future collaboration demands

[Pumareja et al. 2004]. A toolset that encapsulates low-level complexities propitiates the investigation of the interaction through prototyping. With this toolset, a large number of applications may be created, some commercially, others for fun and others by students as class assignments. This favors diversity and creativity in development and enables technological advances through investigation of empirical rules.

Component-based development has being largely used in the literature to address these problems [Roseman & Greenberg, 1992][Slagter & Biemans, 2000][Wulf et al., 2008]. Components make it possible to deal with collaboration design at a high level. Components encapsulate technical issues and business rules, provided by specialists and obtained by experimentation and reused in many diverse situations [Szyperski, 2002]. This article proposes the use of component kits organized and conceived based on the 3C collaboration model [Ellis et al, 1991][Fuks et al., 2007], where collaboration is analyzed from communication, coordination and cooperation dimensions. Software engineers are provided with a specific componentized infrastructure for the groupware domain based on a collaboration model, in order to instrument the construction and maintenance of extensible and adaptable collaborative systems, so as to deal with the collaboration project in a high level. Computer assistant tools and encapsulated building blocks enhance software developers' creativity, resulting in novel and useful products [Winograd, 1996].

2. Groupware workbench architecture

There are recurrent services found in groupware environments. For example, most systems provide forums, chat, agenda, activity reports, questionnaires, task management, voting, repository and links management. Each service may be seen as an independent unit and each group that uses a collaborative environment has specific communication, coordination and cooperation needs. These characteristics are appropriate for the application of component-based development techniques, where the collaborative services are groupware components. Based on component kits organized according to the 3C model, the developer composes an application to provide support for the collaboration dynamics, also modeled according to the 3C model.

The same analysis applied to the system and its services is applied to services and their functionalities, which are recurrent. For example, several services within an environment reuse message categorization and evaluation, permission control, among others. Encapsulating these functionalities into components would also allow other developers to (re)use them in their projects. It also makes it possible to evolve, adjust and build services by varying and reconfiguring collaboration components.

Following this approach, componentization is proposed here in two levels: in the composition of the collaborative system and in the composition of each collaborative service. The developer of a tool selects 3C components that meet the characteristics of support to collaboration regarding the use of the tool in support to the collaborative activity, prototyping and refining support to collaboration iteratively. As the tool changes the way people work, and when new working needs arise, the tool's evolution capability becomes crucial for the success of its use. The iterative process facilitates the development of a more flexible system that is better adjusted to the needs of its users. A family of applications is generated from a component kit, using different combinations and sometimes developing other components on demand.

A component framework is used for either component type (service, collaboration), allowing the peculiarities of each one to be met. Services are plugged into the *Service Component Framework* for the composition of groupware environments, and collaboration components are plugged into the *Collaboration Component Framework* for the composition of services. Component frameworks are responsible for handling the installation, removal, updating, deactivation, localization, deployment, configuration and monitoring of components. The *Service Component Framework* manages the instances of the services and their links to the corresponding collaboration components. Some collaboration components offer interface widgets to facilitate interface construction for the services.

A component's deployment is made by inserting the encapsulated component into the deployment directory. As each component framework has its own deployment directory, may exist several instances of component frameworks in multiple applications. The component is encapsulated in a directory or a ZIP file. The administrator installs, updates or removes services, including, substituting and excluding files. When initialized, the component framework reads the file system and detects changes in the group of components. The component framework also verifies eventual dependences of components and versions.

Customization is the ability to adapt a component before its installation or use, normally to specialize its behavior. In the customization by alteration of properties, a descriptor file is used to configure the component [Szyperski, 2002]. The descriptor file supplies information on package contents, external dependences and component configurations. This file is used by the execution infrastructure to install and configure the component. In the component deployment, the developer of the application may modify specific component configurations. Hence, when deploying multiple times the same component, it is possible to have different configuration for each one. Dependences among collaboration components are described in the configuration file and are handled by the component framework.

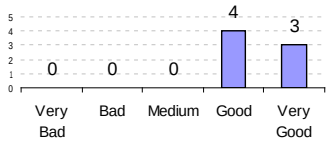
Component instance is the set of data that represents a specific execution of the component. The same service may have many independent instances. For example, in a learning environment, an independent component instance is created for each course or class that uses the same service. The *Service Component Framework* manages the component instances and keeps the current state of each of them, enabling later restoration.

3. Evaluation

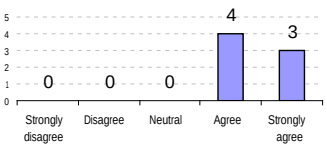
Undergraduate and graduate students are expected to use the workbench to create groupware applications and develop new collaboration services and components. The code produced by the learners is observed and questionnaires and interviews are conducted to generate feedback on the use of the workbench. In this section the results of the developments performed by six master's and one doctorate student, all belonging to the same class, are analyzed. All these students can be considered software engineers. Five of them, besides attending a graduate computer science course, act professionally in the area. None of them has previous experience in developing groupware.

Of the seven developers, six did not have previous experience in Java language, a fact that, according to them, made learning the workbench very difficult. Some of them commented

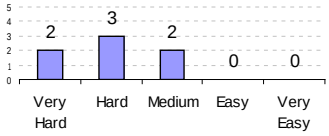
that they spent most of their time studying the basic technologies (JSP, Servlets, MVC and some Java libraries). Neither did these developers have any experience nor knowledge in component based development. As illustrated in Figure 1, the developers evaluated the idea of using 3C model based components for the composition of collaborative systems as being either good or very good, and agreed that the collaboration dynamics activities (collaboration requisites), which are modeled from the 3C model, are mapped for computational support, which is implemented from collaboration components and services. They considered the learning curve of the components workbench from average to very hard. However, they agreed that after the initial learning (a high cost in any components based system), the workbench becomes a powerful tool in component-based groupware development.



How do you evaluate the idea of using 3C model based components for composition of collaborative systems?



Collaboration dynamics activities, which are modeled from the 3C model, are mapped for computational support, which is implemented from collaboration components and services.



How do you evaluate the learning curve of the components workbench?

Figure 1. Evaluation of the workbench

The evaluations of the workbench regarding some characteristics normally considered benefits in the component-based development literature are presented in Table 3. The workbench was best evaluated with regards to prototyping and experimentation of computational support to collaboration, maintainability of the resulting collaborative systems and degree of reuse provided. The developers agreed that the workbench propitiates groupware prototyping.

	Very bad	Bad	Average	Good	Very good
Application composition			1	6	
Development of new components			4	3	
Customization of components			1	4	2
Low level complexity encapsulation			1	5	1
Business knowledge encapsulation			3	4	
Prototyping and experimentation				6	1
Group and parallel development			2	5	
Replacement of components by others			2	3	2
Necessity to know internal implementation detais		2	2	2	1
Maintainability of the resulting collaborative systems				4	3
Degree of reuse provided by the workbench				3	4

Table 3. Components workbench evaluation

The developers were requested to implement a collaborative system, extending the functionalities normally found in similar applications. One of them implemented a shared calendar system, developing the Calendar service and the collaboration components AgendaMgr, EventMgr, TaskMgr and InvitationMgr. Each of these components presents an average of 5 classes and a total of 166 lines of code. Two developers adapted external tools, a Wiki and a chat, to the structure of the workbench. Another one developed components to handle learning objects and file uploading. Two students developed components for their research subjects.

4. Conclusion

Groupware is evolutionary. The composition and the characteristics of workgroups and executed tasks change over time. As a group learns, affinities and conflicts appear, while people enter and leave, causing the group to change continually. In addition, due to the complexity of human interactions, the area is highly interdisciplinary. Hence, low-level complexities should be encapsulated and the design of interaction through prototyping should be better supported [Moggridge, 2006]. Componentization provides the capacity to set up and evolve a specific work environment, selecting and configuring a set of collaborative tools, in order to prototype the support to collaboration. In the proposed approach, groupware composition is made by placing files in the file system and changing XML descriptor files.

It is hoped that with the Groupware Workbench researchers, developers, and undergraduate and graduate students innovate and experiment with the development of collaboration support systems, leading to reuses and integration among different systems. In the evaluation, after the initial understanding of the workbench, which was painstaking during the familiarization exercise, it took developers close to 2 weeks to implement their tools. They reported that the workbench truly facilitated the construction of collaborative systems in providing a series of ready-made components and functionalities provided by the component frameworks.

ACKNOWLEDGEMENT

This work is partially funded by FAPES, with grants of Project 38874849/2007. Hugo Fuks receive grant from CNPq nº 301917/2005-1 also from FAPERJ project Cientistas do Nosso Estado.

REFERENCES

Ellis, C.A., Gibbs, S.J. & Rein, G.L. (1991) Groupware - Some Issues and Experiences. Communications of the ACM, Vol. 34, No. 1, pp. 38-58.

Engelbart, D. & English, W. (1968) Research Center for Augmenting Human Intellect, Proc. Fall Joint Computing Conference, AFIPS Press, 395-410

Greenberg, S. (2007) “Toolkits and Interface Creativity”, Journal of Multimedia Tools and Applications, v.32, Num 2, Special Issue on Groupware and Multimedia, Kluwer, pp. 139-159.

Tse, E. & Greenberg, S. (2004) “Rapidly prototyping Single Display Groupware through the SDGToolkit”. 5th Conference on Australasian User interface, p.101-110.

Pumareja, D., Sikkel, K. & Wieringa, R. (2004) “Understanding the dynamics of requirements evolution: a comparative case study of groupware implementation”, 10th Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2004), Essener Informatik Beiträge 9, pp. 177-194.

Roseman, M. & Greenberg, S. (1992). GroupKit: A groupware toolkit for building real-time conferencing applications. Proceedings of the ACM CSCW, Toronto, Canada, 43-50.

Wulf, V., Pipek, V. & Won, M. (2008) “Component-based tailorability: Enabling highly flexible software applications”, International Journal of Human-Computer Studies, V. 66, Issue 1, January 2008, pp 1-22

Szyperski, C. (2002), Component Software: Beyond Object-Oriented Programming. ACM Press/Addison-Wesley, USA, 2002. ISBN 0201745720

Fuks, H., Raposo, A., Gerosa, M.A., Pimentel, M. & Lucena, C.J.P. (2007) “The 3C Collaboration Model” in: The Encyclopedia of E-Collaboration, Ned Kock (org), pp. 637-644.

Winograd, T. (1996). Bring Design to Software', Addison Wesley.

Moggridge, B. (2006) Designing Interactions, The MIT Press, ISBN 0262134748.