Object Components for Cooperation
A Highly Customizable Tutoring-System with Java-Beans

Jakob Hummes and Bernard Merialdo
Institute EURECOM, France

Jakob Hummes

Institut EURECOM
2229, route des Cretes B.P. 193
06904 Sophia Antipolis Cedex
FRANCE
+33 4.93.00.26.70
hummes@eurecom.fr

Bernard Merialdo

Institut EURECOM
2229, route des Cretes B.P. 193
06904 Sophia Antipolis Cedex
FRANCE
+33 4.93.00.26.29
merialdo@eurecom.fr

Submission to:
ECSCW'97 Workshop on Object Oriented Groupware Platforms
September 7, 1997, Lancaster, UK

Submission category: position paper

Primary contact person: *Jakob Hummes*

{this page is left blank intentionally}

Object Components for Cooperation
A Highly Customizable Tutoring-System with JavaBeans

Jakob Hummes and Bernard Merialdo
Institute EURECOM, France
{hummes,merialdo}@eurecom.fr

Abstract

This paper introduces five hypotheses for designing successful groupware. The hypotheses can be met by using a component model and add groupware-specific components to form a groupware platform on top of that model.
We discuss the strengths and weaknesses of JavaBeans as underlying model. A scenario from the remote teaching domain highlights the usefulness of a component-based approach.
From the hypotheses and the scenarios we draw the conclusion, what still needs to be investigated and developed to create a component-based platform for highly customizable, tailorable and extendable groupware.

1. Introduction

In contrast to stand-alone applications, most applications for cooperation are not very customizable, neither at their user-interfaces nor at their cooperation structure. While the need of radically customizable and tailorable tools for CSCW is known for several years [1,3], only some prototypes exist [1,4,5] that are in general not interoperable with existing applications.

On the other side, object-oriented analysis and design promises to create highly reusable software by modeling the real world in a more natural way. With the outcome of the standardized common object request broker architecture (CORBA) and its standard wire protocol IIOP by the Open Management Group (OMG), applications by different vendors can interact over different object request brokers [6].

While CORBA supports multi-language bindings in a heterogeneous environment, the new object-oriented language Java forms a homogeneous programming environment by running on a virtual machine, which abstracts from the actual hardware. Java supports remote method invocation and object migration through serialization. Having the underlying architecture that allows to create distributed object-oriented programs, the next step is to isolate functionality into object components, which can be easily hooked together. Those components are also called business objects in the literature [8,9] and behave in the analogy of « plug and play ». The OMG is currently reviewing the request for proposals on business objects [7] and JavaSoft has launched with JavaBeans [2] a first infrastructure for component objects integrated within Java.

2. Five hypotheses for successful groupware

We believe that the following five hypotheses must be met by successful groupware and thus should also be supported by groupware development tools.

- **(Integratability)** Groupware must not replace standard applications.

In general, users seldom accept to replace their favorite applications (e.g. web browsers, word processors, spreadsheets) to get group support. Instead, groupware may offer separately additional functionality or should encapsulate or be encapsulated within the favorite applications.

- **(Local interoperability)** Groupware must work with component frameworks.

On platforms that support component frameworks the groupware must also be able to understand this component model to support inter-application interoperability. Objects may be dragged and dropped by the user from a stand-alone application to a groupware application, and vice-versa.

- **(Distributed interoperability)** Groupware must be executable and interoperable on different platforms.

Most groupware is inherently distributed. Different platforms should be supported to not exclude some members of the group. This also implies that the groupware must be interoperable across platforms. Two different approaches exist to solve the interoperability challenge. A standardized distributed architecture uses an also standardized communications protocol (e.g. Corba), or a virtual machine abstracts from the actual platform (e.g. Java or interpreted languages). A virtual machine also enables cross-platform code without the need of recompiling or platform-specific adaptations.

- **(Maintainability)** Groupware must be easy upgradeable.

Distributed applications need easy versioning control or update functionality to ensure that they can interoperate after a change in one component. Versioning control is offered as a Corba service, while Java supports downloadable classes to address this issue.

- **(Customization support)** Groupware development tools must support all target groups, i.e. the initial programmer, the application designer, the experienced user (power-user), and the end-user.

All mentioned points should be supported by development tools. Groupware specific support should thus be included in existing tools. However, the strong division into software programmer and end-users is more and more disappearing. To gain a higher productivity not only code libraries must be reused, but also whole components that can be assembled by specialized third parties to applications. These applications may then be customized to the user's needs by the user itself.

3. Java Beans as component model for groupware

JavaBeans is the software component model for Java. « A bean is a reusable component that can be manipulated visually in a builder tool » [2]. Beans support therefore introspection by tools, customization and properties, events as wiring mechanism, and persistence. This section shows how JavaBeans as component model correspond to the hypotheses for groupware given in the last section.

Java is today the language for the Internet and supported by the leading Web browser manufactures, as Netscape and Microsoft. So, Java Beans are supported inside applets. Assuming that in future other standard software will eventually include a virtual machine to incorporate Java objects, they will also form a framework. Through the release of the ActiveX bridge it is already possible to include Java Beans as ActiveX components in the framework by Microsoft. Beans can therefore be contained by standard applications and perform inside their specialized group tasks.

While Java Beans may be today incorporated as ActiveX controls in a framework, a framework which is built for beans support is yet not available. Java Beans as component architecture is also still evolving. Rules for nesting beans and their interaction with the surrounding containers are not defined yet, but the specification is announced for the end of this year (aka. Glasgow spec.).

Java's virtual machine forms a uniform platform for all Java programs. Java works well with Corba to support communication with remote Corba objects that may also be implemented in other languages. Additionally, Java-to-Java communication is supported by the remote method invocation (RMI) facility. With RMI it is also possible to ship behavior, i.e. classes and their states.

Since groupware is inherently distributed, automatically update of the remote peers is an issue to ensure the communication also after the communication establisher has upgraded their versions. Corba offers with versioning control at least that older clients may still communicate with upgraded servers, if the interface contains only additional definitions. With the possibility to easily download code within Java (as applets or via RMI), clients can automatically download the new functionality from the server. However, the problem remains how this code can then also meaningfully being inserted into the existing groupware application, if not the whole application is downloaded each time. A well designed component-based application could do the job, but further research is required to specify standard patterns for this usage.

Integrated development environments (IDE) for components are able to support the whole development cycle, from the developer to the end-user. Since groupware must often be adapted to the specific needs of the environment, where it is used, it should be extendable and customizable. IDEs for Java and its component architecture Java Beans already exist and can be used to develop beans for cooperation, assemble them to applications, and customize their properties and interactions visually. However all available IDEs only support the local interactions between Beans, but not interactions that cross the borders of their framework, i.e. Java's virtual machine. To set up distributed components in an IDE, it optimally should support meaning for connecting beans across machine and framework boundaries and expose it visually to the user.

4. Scenarios with component objects for cooperation

We developed some scenarios, where groupware gains through using the component approach. Our scenarios are located in the domain of remote teaching.

In a traditional laboratory course, computer science students are working with an application in front of their terminals. They are guided and supervised by one ore more tutors, who react on questions by the students. Before a student asks a question, he searches the tutor's awareness by hand-raising. The tutor moves then to the student and solves the problem cooperatively.

In a remote lab course, the students and tutors are spatially separated. Tutoring is ensured by a groupware system. This tutoring system needs a natural means for a hand-raising analogy and for the student-tutor interaction. However, the analogy does not limit the system to perform further tasks that are impossible in a non-computer

supported environment. This section will give some scenarios to show the strengths of a component architecture for the problem to call a tutor.

4.1 Base scenario: Call a tutor

The base scenario consists of two groups : the students and the tutors. The basic facility is delivered by a student bean to call a tutor, which resides in an application or related compound document (e.g. an HTML page), and a bean that informs a tutor about a help request inside the tutor's control application.

The student bean to call a tutor consists itself of beans, which form the human-computer interface, process the user's input, and sends the help request to the tutors. The user's interface bean may consist of a simple *CallTutor* button, which fires an event, when it is pressed. The bean that processes the help request has registered its interest in that event and sends an help request event to a *SendHelpRequest* bean that forwards the event to the group of tutors.

Even in this simple scenario, customization of the student bean (and its contained beans) is feasible. The simplest customization is to change the appearance towards the user, e.g. to change the button's properties (e.g. label, color) or to use another GUI element, which fires the needed event ; this simple tailoring may also be offered to the students. The *SendHelpRequest* bean needs to know, which tutors are available. Therefore all tutors must register at start-up at a group address. The address may be changed by the properties at design-time. By locally implementing the interaction with the remote tutors in a separate bean, it can be exchanged easily to change the communication protocol (e.g. from Java RMI to Corba).

4.2 Extensions

The simple scenario to call a tutor can be extended by inserting new beans that perform additional tasks. Instead of informing all tutors about a help request, a trader can be included to choose a tutor. Additional information may be gathered by specialized beans at the student side and be used to find the best suited tutor and to be preprocessed by a component in the tutor's application.

The trader is a third part of the system, which intercepts the events that are sent to the group of tutors, processes the information on basis of the registered characteristics and availability of the tutors and forward the request to the best suited tutor (or a subgroup).

In general, additional information can be gathered from three different actors : the application or compound document, in which the student bean resides, the student himself, and a repository, where statistics about the application and its usage are stored and periodically updated.

If the embedding application supports an interface to the student bean, to be asked about its current state this information can be used to specialize the query of the trader and also to point the finally contacted tutor to the origin of the student's problem. The bean, which processes the help request, examines the state of the student application before it triggers the *SendHelpRequest* bean. Instead of implementing the means of interrogating the application itself, a new *GetInfo* bean is inserted that offers a standard interface to the student bean ; for each application a different *GetInfo* bean may exist.

The student is also a good information source. A *GetInfo* bean may present the student a questionnaire that is linked with this instance of the student bean. The questionnaire pops up, when the student calls a tutor to specify his question or problem. The questionnaire itself is statically available (i.e. known at design time of the student bean) or may be created on the fly. Instead of being presented each time the student calls a tutor, the student bean can be designed thus that the presentation of the questionnaire is triggered by the tutor to get more information.

Static information, such as the name of the student and of the application, is stored. The *GetInfo* bean may also query a bean that gets periodically events from the application (e.g. when the student has finished a task).

4.3 Mobile code

The scenarios, but also the fact to work with distributed object components, lead to the consideration of mobile code. A trivial example is, if the *student* bean is inserted in HTML pages and downloaded with a server ; then the bean comes as Java applet. A more complex scenario requires the remote installation of an additional bean in the *Student* bean.

The scenario, where a questionnaire is presented to the student, leads also to the following idea : The tutor maintains a set of prepared questionnaires and sends the calling student one of them to get the needed information. Since the questionnaire does not need to consist only of textual data, but may also comprise active GUI elements that may be displayed differently (e.g. language, expertise level), the actual code must be sent to the student and be executed in his environment. The student can then switch the context at run-time.

Another example, where mobile code can be used, is the serialization of the student application and sending it to the tutor for further investigation. This scenario is applicable, if the tutor needs the state of the application and must be able to manipulate it in order to help the student. It can also be used to distribute exams on-line and urge the exam to close itself after the time limit and return to the tutor.

5. Requirements for component based groupware platforms

This paper has introduced hypotheses that should be met by groupware systems to be successful. We showed that JavaBeans is suitable as component platform and developed scenarios built upon this new component model. However, developing and implementing the scenarios, we encountered some limitations that we will discuss in this section.

5.1 Component model and IDE

A component-based groupware platform needs, of course, a component model. This model must be supported by IDEs to support all involved development groups : from the programmer to the end-user.

JavaBeans offer such a component model and IDEs for creating, plugging and customizing beans are commercially available. However, JavaBeans still lacks standardized means for nested components and interaction with the surrounding containers and frameworks.

5.2 Remote introspection

To use components in a not foreseeable way, it is necessary to analyze their capabilities. JavaBeans offer introspection to locally analyze beans.

Since groupware is inherently distributed, it is needed to analyze also remote beans, if they should be integrated in a seamless manner. The connections between remotely located beans do not need to be transparent ; instead the different locations should be visible to the application designer. Supported by an IDE for remote components, the designer can connect the events between the beans across the boundaries of the virtual machines.

A visualized group metaphor for configuring the system (e.g. events from a student to the group of tutors) should also be supported by an IDE for groupware.

5.3 Code shipment

In monolithic groupware applications, all possible interaction types between the group members is known in advance. On a component-based groupware platform, it is also desirable to extend the functionality at runtime, i.e. to distribute a component, which is then inserted correctly in the existing groupware applications.

6. Conclusion and outlook

Driven by our five hypotheses for successful groupware, we found that JavaBeans is a promising component model for stand-alone applications. We argue that a groupware platform can be built on top of this component model and will so inherit the benefits of a widely accepted platform.

Our research strategy is to develop groupware scenarios in the tele-teaching domain, to identify the reusable object components and implement some example scenarios.

From the presented scenarios, the base system for the student and the tutor is currently being implemented. Some of the extensions are also foreseen for implementation. We are using Java RMI to fire events to remote listeners and have developed beans for the group abstraction. However, we are also discussing to use Corba and its services. As example, the trader in the given scenario may be a service built with JavaBeans, but it is also in the discussion to use the Corba trading service. If we choose to build our own trader, then because of the need to implement other functionality than the Corba service specifies.

At the same time we have developed further scenarios, which can reuse some of the implemented beans, but also offer new challenges, as code distribution and object customization at run-time.

7. Acknowledgments

The described work is part of the ACOST research project, which is funded by the research institute CNET Lannion of France Telecom.

References

[1] Dourish, P., *Open implementation and flexibility in CSCW toolkits*. Ph.D. Thesis, University College London, 1996, ftp://cs.ucl.ac.uk/darpa/jpd/dourish-thesis.ps.gz.

[2] JavaSoft, *Java Beans 1.0 API specification,* Sun Microsystems JavaSoft, Mountain View, CA, Oct. 1996, http ://java.sun.com/beans

[3] Malone T.W., K.R. Grant, K-Y Lai, R. Rao, and D. Rosenblitt, 'Semistructured messages are suprisingly useful for computer-supported coordination'. In Irene Greif (ed.), *Computer-supported cooperative work - A book of readings.* Morgan Kaufman, 1988, p. 311-331.

[4] Malone T.W., K-Y Lai, and C. Fry, *Experiments with Oval : a radically tailorable tool for cooperative work* ACM Transactions on Information Systems, 1995, 13(2), p. 175-205.

[5] Mines R.F., J.A. Friesen, and C.L. Yang *DAVE : a plug and play model for distributed multimedia application development,* ACM Multimedia, 1994, (10), p 59-66.

[6] Mowbray, T.J. and R. Zahavi *Essential CORBA - Systems integration using distributed objects.* Wiley, NY, 1995.

[7] Object Management Group *Common facilities RFP-4 : Common Business Objects and Business Object Facility* Object Management Group, Framingham, MA, USA, 1996, http ://www.omg.org/library/schedule/CF_RFP4.htm

[8] Orfali, R., D. Harkey and J. Edwards, *The essential distributed objects survival guide*. Wiley, NY, 1996.

[9] Sims, O. *Business objects - Delivering cooperative objects for client-server* McGraw-Hill, 1994