Software Components for Cooperation : a Solution for the « Get Help » Problem

Jakob Hummes, Arnd Kohrs, Bernard Merialdo
Institute EURECOM, 06904 Sophia-Antipolis
{hummes, kohrs, merialdo}@eurecom.fr

ABSTRACT

In this paper, we study the usage of software components to build cooperative applications. We focus on a situation arising within tele-tutoring applications: the « Get Help » problem, where students request assistance from a tutor (among the available tutors) during a laboratory exercise. We analyze the case where students and tutors are face-to-face, and the case where they are remote users of a distributed system. The Java Beans component model is used to create and modify distributed applications, which support this interaction. We extend this environment with beans that support group communication and show how beans can be visually combined to create a solution for « Get Help » situations in different contexts.

KEYWORDS
tele-teaching, group communication, customization, JavaBeans.

1. INTRODUCTION

The need of radically customizable and tailorable tools and applications for CSCW is known for several years (Dourish, 1996 ; Malone 1988). In the domain of tele-teaching customization is specially interesting, because teaching scenarios differ from one scenario to the next, while they have similarities. These differences can in general not all be included as options in monolithic computer-supported teaching applications.

Software engineering has recently introduced the component technology, where objects are assembled to independent subsystems with standardized interfaces. These components can be easily assembled to create new components or applications. Components for distributed computing are often called business objects (Sims, 1994) and behave in the analogy of « plug and play ». The standardized interfaces of components allow to customize and interconnect components with the help of visual builder tools and to be plugged into frameworks.

One can see the impact of the component technology by the effort taken by standardizing organizations and leading manufacturers to define and establish component models. The Open Management Group (1996) is currently reviewing the request for proposals on business objects and for a CORBA component model, Microsoft has established ActiveX (Chappell, 1997), and JavaSoft (1996) has launched with JavaBeans a component model for Java.

This paper proposes to use the benefits of component technology to build highly customizable and reusable groupware. The « get help » problem serves as example for this approach.

2. THE « GET HELP » PROBLEM

The « get help » problem manifests itself in tutoring situations, as in laboratory courses or in hot-line situations. In the traditional teaching environment of a classroom, students are assigned exercises during a laboratory course. The students are solving their tasks on computers, while one or more tutors are in the classroom to instruct, supervise and help the students, if they need assistance. When one of the students wants assistance, he informs a tutor by raising his hand. Each tutor becomes aware

about the student's wish. One tutor then moves to the student, offers his help and they solve the problem cooperatively.
When this situation is to become computer-supported, some questions arise.

- What is known about the student's problem ?

  In a remote laboratory course, where tutors and students are spatially separated, an analogy for the hand raising must be found. Additional information can be collected from the application.
- Which tutor is informed about the help request ?

  A computer-supported lab-course could inform all tutors about a request and inform the others, when one tutor has decided to help this student. If more information is available, the system could assign a tutor, who is the best suited to solve that particular problem or try to balance the work between several tutors.
- How can the tutor and the student communicate ?

  In the classroom scenario, the communication is evident : The tutor moves to the student ; they communicate face-to-face with all possible interactions. In the remote tutoring scenario, the interaction must be computer supported. The interaction can include symmetric communication forms, as audio, video, and text-based chat, but also asymmetric forms as messages. An additional form of interaction would share the application and use tele-pointers.

These questions and the variety of the possible answers give an example that a monolithic application that wants to support all degrees of freedom is not applicable. This paper presents an approach, how the degrees of freedom in the « get help » problem can be preserved by using components, which can be assembled and customized easily to be incorporated in such a tutoring system.

3. COMPONENTS AND EVENTS

The JavaBeans component model uses events as a configurable means for the exchange of messages between components. Components and events can be manipulated visually in builder tools. In order to preserve the configurability, events are used to update distributed components with current states of the tutoring system. Tutoring components are themselves aggregated from finer grained components.
The component that interacts with the student is called *StudentBean*, the component for the tutor consequently *TutorBean* ; optionally a *TraderBean* can be integrated in the system to offer a more sophisticated way to find a well suited tutor.
The following events are sufficient to implement the message exchange in a distributed tutoring system, which offers the needed degrees of freedom to support various customizations, as including different cooperation components. For this reason the events may additionally carry arbitrary information.
The *NeedTutor* event is emitted by the *StudentBean*. It carries information to identify the student, a call-back address to set up the cooperation, and an arbitrary object that describes the student's problem. The description can be supplied by the student directly or derived by the state of the student's tele-teaching application. A *Cancel* event revokes the request.
The *OfferTutoring* event is sent by a *TutorBean*. It contains information about the identity of the tutor and the help method. The help method is used to lance the facility of cooperation support, which can include a textual chat, audio/video conference, or any other method. The *Finish* event indicates that the cooperation phase is over.
In order to support the integration of other components, events for registering and unregistering are designed. They can be used locally to

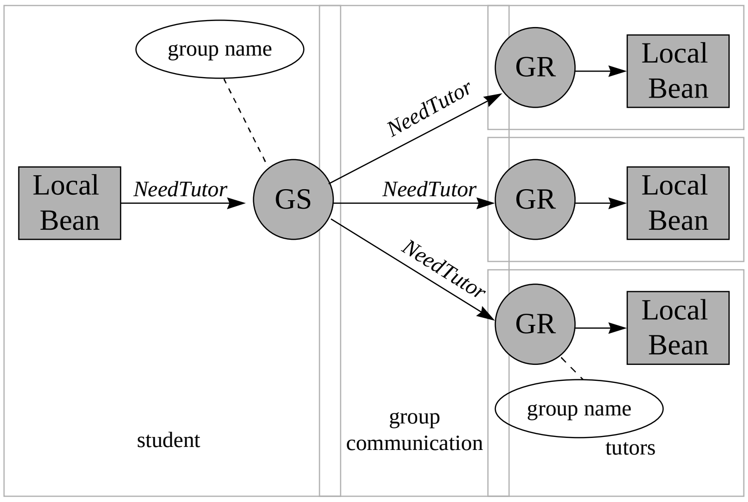

Figure 1 : Beans for group communication (GS : GroupSender, GR : GroupReceiver).

set the properties of the *NeedTutor* and *OfferTutoring* event, but also be distributed to a *TraderBean* that implements a strategy to find the best suited tutor. To offer full customization support the *Register* event can carry a strategy object, which is taken by the trader to fulfill its task.
Since the JavaBeans component model defines only the interaction between beans in the same virtual machine we developed group communication beans, which act as access point to distribute an event to a group. The group communication beans expose the event model to the developer for remote event communication. Two beans are necessary : The *GroupSender* forwards an event to all *GroupReceivers*, which are configured with the same group name (Figure 1). The group name is a property of the beans and can easily set within visual builder tools for beans and the events can be visually connected to and from these beans.

4. APPLICATIONS FOR THE TUTORING SYSTEM

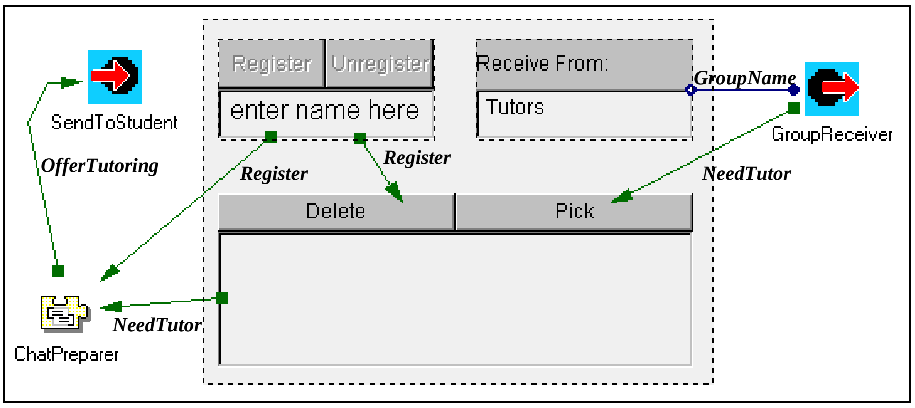

Figure 2 : Simple tutor application in the IDE.

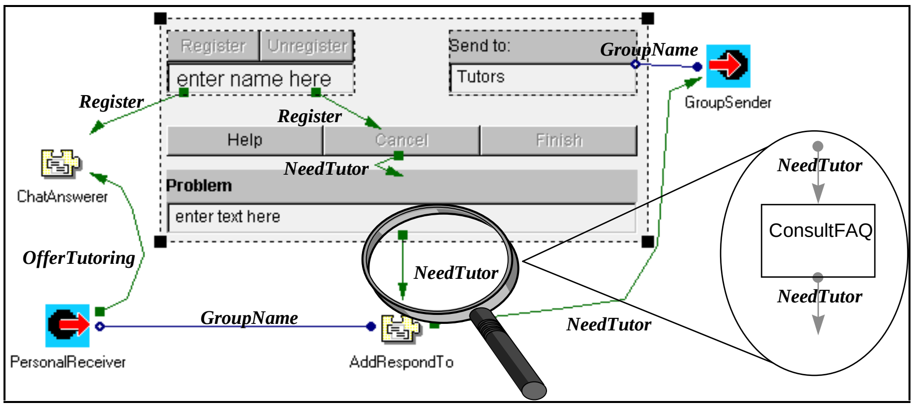

Figure 3 : Student application in the IDE.

The tutor interface to the tutoring system could be assembled as shown in Figure 2, the student application visually assembled as in Figure 3. Both examples contain visible, i.e. graphical user-interface, beans and invisible beans that are only visible in the builder tool at design-time. The beans named *GroupSender* and *SendToStudent*, are configured *GroupSender* beans ; *PersonalReceiver* and *GroupReceiver* are configured *GroupReceiver* beans. In these examples the *NeedTutor* event is sent to a group, which can be reconfigured at run-time, and defaults to the name « Tutors ».
The IDE allows to easily modify an existing application: for example, if we want a FAQ to be searched automatically for already answered questions before sending the question to the tutor, we simply have to insert a new bean *ConsultFAQ*, as indicated in Figure 3.
When a tutor picks a request for help, the *NeedTutor* event is forwarded to a *ChatPreparer* bean, which issues an *OfferTutoring* event. The *ChatPreparer* in the tutor application and the ChatAnswerer in the student application manage the configured cooperation facility : a (here not shown) chat bean for a text-based conference. To change the cooperation mechanism only these beans must be exchanged. We have also implemented a bean pair for asymmetric cooperation. The *StudentBean* can be further extended to pass a help request to the tutor only in the case, if the answer is not already given in the FAQ.
Note that only a few connections and beans need to be assembled to include the beans for the « get help » problem in an application. In order to put a trader into the system only other group names for the sender and receiver must be given and a strategy object must be configured with the register event.
We used the developed beans to support a « get help » facility in a tele-exam, where exams are

distributed on-line from the tutor to the students. The tutor is additionally supported with a facility to generate a FAQ on the WWW from the student questions, which is straightforward, since the information can be retrieved from the chat beans easily.
Alternatively, the FAQ can be queried automatically by a configuration as suggested in Figure 3. Then a request is filtered and only sent to a tutor, if the question is not yet answered in the FAQ. As filtering strategy a simple keyword analysis could be taken.

5. CONCLUSION

The presented work shows a generic computer-supported approach to solve the « get help » problem : How can users call for help from other users. The problem is found in a variety of tutoring situations, as in laboratory courses in a university, but also in hot-line services. The actual cooperation that takes place after the requester has found a tutor is not part of the problem ; however, a generic solution must provide the possibilities to plug in different forms of cooperation tools.
The paper argues by looking at tele-teaching scenarios that a monolithic application cannot offer all possible options. The suggested solution bases on software component technology and we provide Java beans, which are visually customizable by off-the-shelf IDEs. The JavaBeans event model is extended by beans for group communication to pass remote events.
The shown examples underline that the approach can be inserted easily in tele-teaching applications also by non-programmers using a visual builder tool. Migrating from a tutoring system with cooperation support for a symmetric chat to an asymmetric cooperation by transmitting a file requires only the exchange of two beans. More sophisticated strategies to find a specialized tutor is supported by a trader. The beans for the « get help » problem were integrated into a tele-exam in our institute to show the inter-working with other components.
The presented beans solve one problem in tutoring situations. Their power will be shown, when more applications use them and more sophisticated beans for cooperation support will exist.

7. ACKNOLEDGEMENTS

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

8. REFERENCES

Buschmann, Frank, Regine Meunier, Hans Rohnert, Peter Sommerland, and Michael Stal. *A System of Patterns.* John Wiley & Sons, 1996.

Chappell, David and David S. Linthicum. 'ActiveX demystified' in *Byte*, pp. 56-64, September 1997.

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.

JavaSoft, *Java Beans 1.0 API specification,* JavaSoft, Oct. 1996, http://java.sun.com/beans

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.

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

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

9. RESUMEE

Dans cet article, nous proposons l'utilisation d'un modèle de composants logiciels pour la construction d'applications coopératives. Nous étudions un scénario issu d'une application de télé-enseignement, celui de la demande d'assistance. Nous montrons comment l'environnement Java Beans, grâce à une extension de communication de groupe que nous avons développée, permet de créer une application distribuée répondant au scénario. De plus, l'environnement de programmation visuel permet facilement de générer des variantes selon les différentes fonctionnalités souhaitées.