Questionnaires:
a Framework using Mobile Code for Component-Based Tele-Exams

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

Abstract

Applications for computer supported cooperative work can gain from component models and frameworks. The framework for "questionnaires", which is described in this paper, offers a pattern to distribute artifacts to a group of receivers. We use mobile code to ensure highest flexibility and provide hooks to be able to collaborate with local applications on the receiver side. The questionnaire framework is not only interesting for tele-exams, which are described in detail in this paper, but can also be used to pass artifacts around in a workflow system.

Our approach supports the whole life-cycle of questionnaires. Standard tools support the design of the questions and their assembling into a questionnaire using components. A questionnaire is distributed to all students in an exam and recollected at the end of the exam. The application for the teacher comprises a component to automatically evaluate the answered questionnaires and to store them persistently.

1. Introduction

One important domain in computer supported cooperative work (CSCW) is the support of education with computers. This domain is often labeled as tele-teaching, remote education, or distance learning.

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.

The need for adaptable, customizable and tailorable tools and applications for CSCW is known for several years [3, 12]. CSCW toolkits as OVAL [13] or Prospero [3] support tailoring and are implemented as prototypes, but are in general not suited to build "real-world" applications or extending existing frameworks.

Our research is focused on defining reusable frameworks for CSCW and validating the findings in the domain of tele-teaching. The approach is based on the observation that CSCW can benefit from emerging component models for object-oriented languages by defining highly reusable components for cooperative activities. Visual builder tools for component models facilitate the adaptation and customization process to the extent that also end-users can make modifications to component-based applications.

In order to take advantage of components written for other purposes, but reusable in cooperative settings, we decided to adopt the general purpose language Java and its component model JavaBeans, where components become increasingly available. Java compiles to a platform independent byte-code. This is especially from significance, since CSCW is inherently distributed, and it allows to pass and execute code on remote computers.

In this paper, we use the term "framework" as defined by D'Souza and Wills [5] to describe a generic package that offers a recurring pattern and is extensible by offering "plug-points". By instantiating and mapping the frameworks to real problems, the resulting components can be viewed as cooperative business objects [19].

The next section will draw the requirements for computer supported tele-exams from the identified life-cycle of questionnaires. Since our approach relies on a component architecture, the third section will introduce JavaBeans and present beans for an already developed framework for remote tutoring. The design of the questionnaire framework will be discussed in greater detail in section 4. Our approach supports all phases in the life of a questionnaire. Subsequently we will show some composed applications as examples. The paper will then relate our approach to other relevant publications and finally conclude with an outline on how further work fits into this approach.

2. Requirements for Tele-Exams

Exams are held in all educational environments to control the progress of the learners. Exams can differ not only by their contents, but also by their purpose and how the questions are being asked. This paper will focus on written exams, which are distributed at the same time to the learners and must be returned before a certain amount of time has been elapsed.

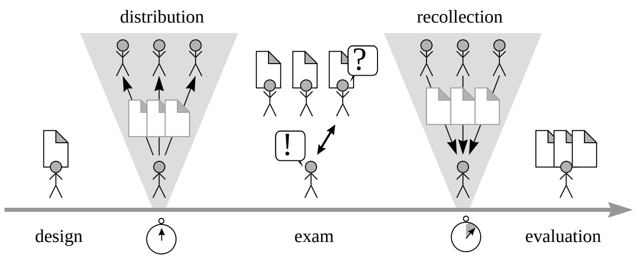


Figure 1. Life cycle of a questionnaire.

Figure 1 shows a typical life-cycle for questionnaires in an exam; the figure holds for traditional paper-based exams as for tele-exams. A professor designs the questions for the exam. The resulting questionnaire is then copied and distributed to all students at the beginning of the exam. The students are filling out the questionnaires. Should they have questions they may ask a tutor for clarification. The exam must be written in a given time; when the period has elapsed all filled questionnaires are recollected. The professor evaluates each exam to grade the students afterwards. Finally all questionnaires are stored persistently.

On-line exams are useful to support spatially dispersed people in remote education scenarios. On-line exams differ from traditional exams also in the way that the exams can use the computer to offer capabilities, which are not available in traditional paper-based exams. Using questionnaires that contain active code add to this advantage more features as it would be possible in paper-based exams. The questionnaires offer the students a better and more intuitive user-interface and let them choose properties, such as the language for presenting the questions. Animations can be delivered as well to help better understanding; an example would be a rotateable 3D view on a molecule for a chemistry course.

In order to support exams by computers, not only their distribution must be solved, but the professor should be given an environment to easily design the artifacts and to evaluate and store the results after recollecting the answers. Consequently, our approach addresses the following requirements:

- The creation of new questionnaires is supported by an integrated development environment (IDE) and a collection of pre-manufactured question types.
- The questionnaires are distributed within the existing tele-teaching framework and plug themselves automatically in the student application.
- The student uses the user-interface of the questionnaire to enter the answers; the interface presents itself individually according to the student's preferences and can be changed at runtime.
- The filled-in questionnaire returns automatically or on request to the tutor.
- The tutor application holds the returned questionnaires and can browse them. A component offers automatic evaluation for special question types like multiple choice questions.
- Questionnaires can be stored persistently to retrieve them later.

3. The Component Approach

Our approach to get a highly customizable tele-teaching system uses the component model JavaBeans for the object-oriented programming language Java. The system benefits from the already recognized features of Java, which makes this language to a de-facto language for the Internet: compiled Java classes are executed in a virtual machine, which abstracts from the actually used platform; distributed environments for Java do not only allow to call remote methods, but also to transmit whole objects consisting of state and behavior.

The component model JavaBeans adds a standard way to manipulate Java classes that conform to the Beans specification within integrated development environments (IDEs).

3.1. Component model: JavaBeans

A component is an independent "unit of software that encapsulates its design and implementation and offers interfaces to the outside, by which it may composed with other components to form a larger whole" [5]. Component models, as JavaBeans, use an event model to facilitate component compositions.

JavaBeans is the component standard for Java. The specification for JavaBeans outlines that "a Java Bean is a reusable software component that can be manipulated visually in a builder tool" [20]. Beans are Java classes that follow so called design patterns to let builder tools introspect a bean and to let a bean being self-descriptive. The standard distinguishes two extraordinary states in the life-cycle of a bean: A bean can be manipulated in a builder tool at design-time or behaves like an ordinary object during run-time.

During operation a bean changes its state. Internal state changes can be made visible to the outside through the event mechanism. It consists of registration-methods, handler-methods and event-objects. The event-mechanism allows the coupling of state transitions of a bean with an action in another bean. The Beans specification describes how the coupling has to be implemented, thus it can be automated through an IDE.

Properties reflect accessible state of a bean. Beans can be customized during design-time by altering bean properties. Properties can be bound, thus special events are fired upon property changes at run-time, which can trigger actions in other beans.

3.2. Customization and tailoring support

Properties and events can be manipulated within visual builder tools. The JavaBeans standard offers also additional associated classes for each bean, which include special customizers and property editors to support a more intuitive interaction with the developer. The standard describes further, how persistence is achieved for the customized beans.

The activity of tailoring is normally defined as an end-user customization process during run-time, while customization at design-time is regarded as a programming process [17]. IDEs that use the capabilities of Java to load code at run-time (such as Visual Age for Java) shorten this distance, by allowing design changes being reflected in the running application, when the class is used. The here presented approach restricts itself however to customization at design-time, but we argue that design-time customization is common and powerful enough to support different tele-teaching scenarios, if the frameworks and components are designed for that goal.

The component-based approach together with visual integrated development environments (IDEs) directly supports our goal to be able to customize an existing application and to be able to build new similar applications by reusing the components. Beans allow even non-sophisticated Java-programmers to customize applications in an intuitive way. The easy grasp is achieved by the use of graphical and form-based editors within the IDEs.

3.3. Mobile Code

The ability to download platform-independent applets and execute them locally and safely in browsers is one of the main reasons of the widespread acceptance of Java. In contrast to Java applets, where code is downloaded by the browsers and then started by calling an initializing method, transmitting a configured bean needs also to transfer the current state of this object. The serialization API supports migration of stateful objects.

Transporting a component over a network, however, is only one issue. To be meaningful, the component must be plugged in at arrival to collaborate with the framework consisting of other components. The event model of JavaBeans helps defining such “plug-points”. The arriving component registers for the events from the local components; the local components also register their interests with the mobile component.

The possible plug-points are hereby defined in Java interfaces. As long as the components conform with the interfaces they can provide different implementations and internal behavior; the components can still collaborate.

3.4. Beans for group-communication

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 or to subscribe to a message from a group. The group communication beans expose the Java event model visually 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. The design for group communication follows the publisher/subscriber pattern [2], the group name corresponds to the subscription. The group name is a property of the beans and can so easily set within a visual builder tool for beans at design-time or can be exposed as option in the user-interface to allow changes at run-time.

The beans for group communication are designed on a higher level than the actually used distributed system for the implementation. Our implementation uses the agent-enhanced ORB for Java of Voyager [18]. The design of the beans guarantees that only the interface to the underlying communication system must be developed in order to exchange the distributed system. The higher level bean is not affected thus that the communication system can be exchanged without affecting already configured systems that use these group communication beans.

3.5. Frameworks for Tele-Teaching

Tele-teaching frameworks may support the basic requirements for various educational settings. Each teacher, however, will require special settings for a course or exam. In order to support different and even not yet known settings the system must be highly adaptable. The components for tele-teaching, which we have implemented within our project, are designed to be customizable for experienced users, such as teachers. They offer comprehensible interfaces and do not overwhelm a teacher with functionality.

We developed basic frameworks for tele-teaching scenarios. One of these frameworks, which solves the "get help" problem [9], is also from relevance in tele-exams. For the this framework, we developed components that allow students to request help from one or more tutors. The framework uses the described beans for group-communication. How the student and the tutor actually interact is not defined within the framework. Instead, different beans supporting the desired interaction can be inserted at designed plug-points. Currently beans are implemented that offer a textual chat and the possibility to send an artifact, e.g. answers to frequently asked questions (FAQ).

One implementation for a laboratory course uses the "get help" framework and can be combined with tele-exams to control the learning progress of the students by asking the students to fill out a questionnaire after the lab course. On the other side, a tele-exam can include the "get help" framework to allow students to ask for clarification.

4. Questionnaires

The design of the questionnaires and the applications for the students and teacher are based on the requirements for tele-exams as discussed in section 2. Our questionnaire framework supports exams during the whole life-cycle, from the design of a questionnaire, over the actually held exam, to the possibility of automatically evaluate the answers. This section describes the phases in greater detail.

4.1. Design phase

For the tele-exam, the assembling of question-beans to questionnaires needs to be as simple as possible. Special customizers for all offered question-beans ensure that a composition at design-time can be done with drag-and-drop.

The professor needs only to drag and drop questions within an IDE in a questionnaire container. The questions can be customized visually in a WYSIWYG fashion. The connections for the interactions between the questions and the questionnaire are done automatically. The resulting questionnaire and questions implement all needed interfaces to be plugged into the student application and for the final evaluation by a master copy of the questionnaire together with the right answers (see figure 2).

4.2. Distribution phase

A questionnaire with the contained questions is for the system an ordinary Java object. Also a blank questionnaire contains state (e.g. text, animations, timer). So the distribution takes advantage of the capability of Java to transmit

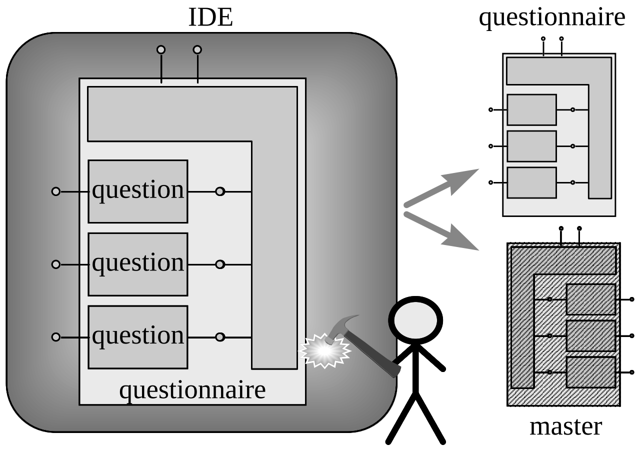


Figure 2. Composition of a questionnaire.

code. The questionnaire and the contained questions implement the needed interfaces to be plug-able in the applications for the students and the teacher. Other beans could have been used also by the teacher, as long as they conform with the interfaces.

The introduced beans for group communication are used to transport events between distributed parts of the system. The questionnaire is sent to all students at the beginning of the exam. Figure 3 illustrates how a questionnaire is distributed. The responsible component in the tutor application (here: a questionnaire manager, see also figure 5) signals a GroupSender bean to send the questionnaire to the group, e.g. all students. The GroupSender serializes the questionnaire and puts it, encapsulated in an event, onto the underlying communication system.

All GroupReceivers that are configured to listen on this event, receive the serialized questionnaire, de-serialize it, and notify the responsible components. In a student application, the questionnaire control registers its interest in some offered events from the questionnaire, and the questionnaire registers itself for events from the control. After the registration phase, the components are plugged, and they may start collaborating.

4.3. Exam phase

During the exam, students are answering the questions, by filling out the questionnaire. In our current design, the questionnaire itself is responsible to offer an appropriate user-interface. Figure 4 shows the collaboration between a questionnaire and the control. The student uses the questionnaire control to manipulate properties of the questionnaire and its questions. Examples are the preferred language for the questions or to lock the questionnaire against unwanted accidental overwriting at browsing the questions.

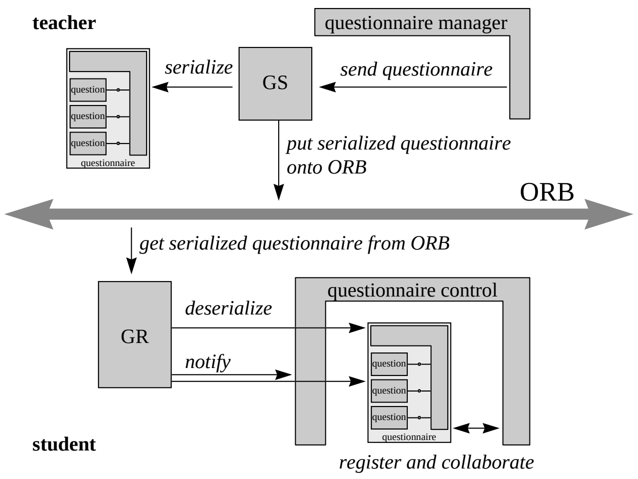


Figure 3. Distribution of questionnaires.

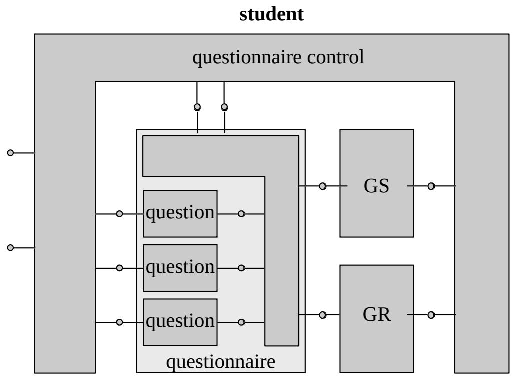


Figure 4. Collaboration between a questionnaire and the student application.

The questionnaire control bean has also the possibility to communicate with other beans. Since also carefully designed exams are sometimes ambiguous, the student needs the possibility to contact a tutor. While the communication with the tutor lays outside the scope of this paper, it is noteworthy to understand that other beans, as beans for the "get help" problem, can collaborate with the control to retrieve more information, e.g. the question, which causes the problem for the student.

An exam is often time constrained. A special bean, which can be inserted into the questionnaire, manages a count-down timer. When the time has elapsed, the timer informs the control that the exam is over and triggers the GroupSender to send the questionnaire back to the configured address, e.g. the professor's application. The functionality for the collection of the exams is the same as described for the distributing case. The questions have changed their states due to the given answers of the students; the serialized questionnaire, which is sent back to the professor reflects the new states, i.e. it contains the answers.

4.4. Evaluation phase

The professor uses a questionnaire manager within his application to evaluate the returned questionnaires, which are plugged in as in described in the distribution case. The manager holds all questionnaires and offers for automatically evaluatable questions an evaluator bean, which is connected with the master copy of the questionnaire as obtained from the design phase (see figure 5), which contains the correct answers.

The evaluation for each questionnaire is passed to a report generator bean. The report can be edited manually by the teacher to include corrections of not automatically evaluated answers, comments and the final grade of the exam.

5. Composed Applications

This section gives an overview how student and tutor applications can be built upon the beans presented in the previous sections. To demonstrate the usefulness of the component approach we outline the combination with other cooperative beans. Also some implemented question beans are introduced that are ready to be inserted into questionnaires.

A student uses the questionnaire control during the exam to receive the questionnaire and set global properties. The questionnaire control bean is combined with the "get help" bean to allow the student to ask the teacher during the exam. Figure 6 shows the questionnaire control, which allows to hide the questions, toggle a lock for editing and select the preferred language. The settings can be made before the questionnaire is received and during the exam. The figure shows also the bean for getting help by a tutor.

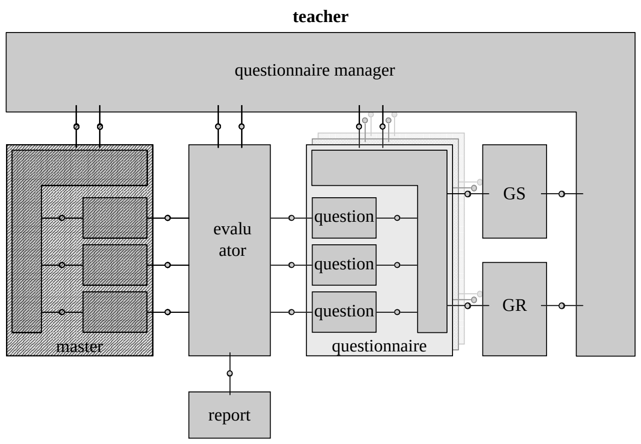


Figure 5. Collaboration between a questionnaire and the professor's application for automatic evaluation.

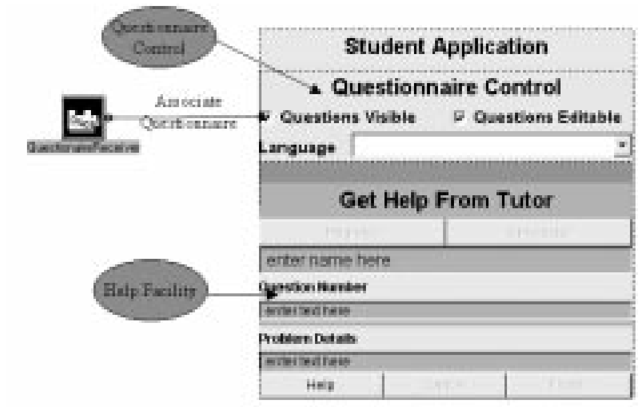


Figure 6. Student application during development.

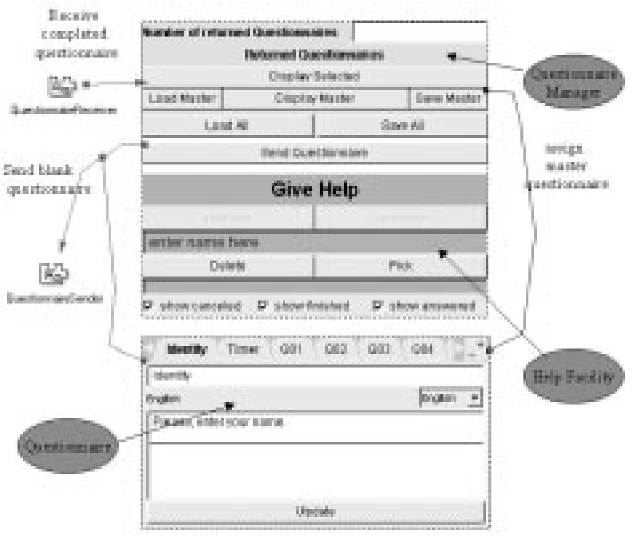


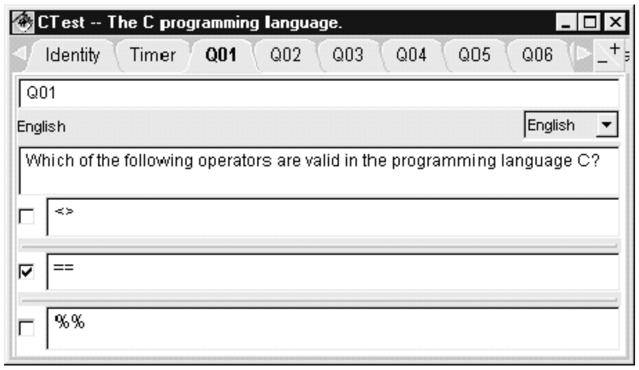
Figure 7. Tutor application during development.

Figure 7 shows the beans for the questionnaire manager combined with the give help facility used by the professor or the tutors. The shown questionnaire manager has a button to send a blank questionnaire to all students. After the answered questionnaires are received, the manager can show them and they can be compared against the master questionnaire. For automatically evaluatable questions an evaluator supports the correction. Questionnaires can be saved persistently and retrieved later.

The help facility shows help requests by the students and can be answered individually. When the tutor chooses a help request a configured communication tool is used; currently beans for chatting and sending back a list of answers of frequently asked questions (FAQ) are implemented. Additionally we support the generation of an HTML FAQ from a chat session during the exam, which can be viewed by the students with standard Web browsers. To guarantee that the viewed FAQ shows always the up-to-date contents, we reused beans developed for the active annotation approach [8].

As example for the questions we use an exam about the C language, how it is held at our institute to test the knowledge of new arriving students. Since Eurecom is an international university in France, the developed questionnaires support the languages English, French and German. The design of the question interfaces however allows to offer as many languages as wanted.

Questions, which implements the interface `Evaluatable` can be evaluated automatically. Currently, two different question types are evaluatable, an



Q01

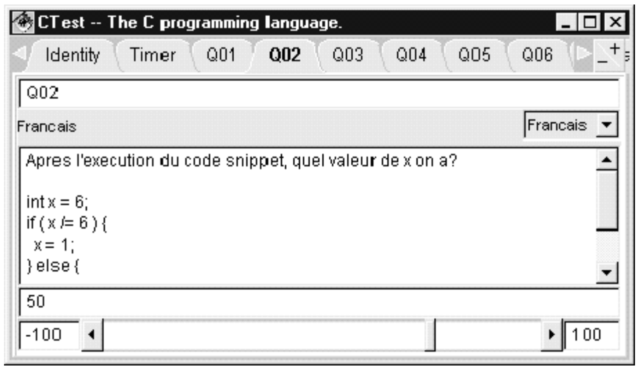
English — English

Which of the following operators are valid in the programming language C?

- [ ] <>
- [✓] ==
- [ ] %%

Figure 8. Multiple Choice question.

integer value question and a multiple choice question. Both types are accompanied also with a user-interface, which supports intuitive input. The multiple choice question type is presented with `CheckBoxes`, as shown in figure 8. A scrollbar supports the choice for a value input, as shown in figure 9. Of course, also other representations could be chosen during design-time. Note that the language can not only be chosen for all questions by using the control, but also individually for each question.



Q02

Francais — Francais

Apres l'execution du code snippet, quel valeur de x on a?

```
int x = 6;
if ( x /= 6 ) {
  x = 1;
} else {
```

50

-100 … 100

Figure 9. Integer value question.

Other beans can be included in a questionnaire as long as they are serializable. They are recognized as question type, if they implement the `Question` interface. A more general question type presents a question and expects a textual answer; this type is however not automatically evaluatable. Specialized beans for user identification and a timer are normally included into an exam to get the student name and to constrain the time of the exam.

6. Related Work

Hiltz [7] and Turoff [21] describe how to set up a virtual classroom and their findings using it consequently. They emphasize that on-line education can be better than traditional classroom education. We hope that our components can serve as a basis to create better exams.

The Assiniboine college offers web based courses that have modules as instructional units [4]. For and after each module, the students are presented short exams to control the learning progress. To prevent cheating, an exam is differently created on the fly out of a pool of questions by cgi-scripts. Our approach could be extended to support a unique creation of exams for each student or to automatically present multiple choices in different order. Moreover, cheating is more complicated within our approach, since beans can not be saved as easily as Web pages; additionally our approach supports to constrain the time of an exam.

The project Nestor [14] uses coarse grained components, or modules, to offer a computer aided learning environment [16]. The need of adaptation to specific learning contexts was recognized and tailoring languages and tools for special purposes, as the course structure, were developed [15]. The collaborative learning and research environment (CLARE) [22] uses a similar approach on top of the extensibility of Emacs. In contrast to these approaches, we are using components in all layers, providing the possibility of adaptation at all levels. By taking JavaBeans, we can also insert third-party beans and use the capabilities of off-the-shelf IDEs.

Franze et al. [6] outline the factors to develop successful courseware. Animations combined with audio and video support showed the best results. They introduce a Java based framework JaTeK, which has similarities to our teleteaching framework. The here presented questionnaires can hold Java based animations; and the approach and implementation could be incorporated in the JaTeK environment. On the other side, it should be possible to integrate components for JaTeK into our environment. We are currently investigating the Java Media Framework to include real-time multimedia support for the student–tutor communication.

As was outlined in the section about our GroupSender and -Receiver components, they do not rely on a specific middleware. An interesting thought would be to combine these components with a framework, which implements already the publisher/subscriber pattern, but offers also collaboration services as described in [1].

Mobile agent systems are already available for Java [11, 18]. We could have used the agent facilities of the actually used middleware of Voyager [18], but we decided against, because we do not see the need of independent agents in the case of questionnaires. By not relying on a specific agent implementation we are free to change the middleware easily. However, we are considering the use of agents for monitoring purposes, as to signal to the tutor the progress of the students. An agent could travel around the students to gather information and present the collected material in periodic updates to the teacher.

7. Conclusion and Further Work

We presented questionnaires containing active code to support tele-exams. We showed that off-the-shelf IDEs can be used to assemble question components within the JavaBeans component model. By supplying additional customizers the creation of a new questionnaire can be done visually by drag and drop question beans from a palette; their texts can be entered in the same step. Active code supports enhanced user-interfaces and offers additional features as language selection. The paper presented the support of the users, teachers and students, for the whole lifecycle of questionnaires, from the design, over the distribution through mobile code, to the automatic evaluation of some question types.

The presented questionnaires are part of a greater teleteaching framework, which is being researched and developed in the ACOST project. Components developed for tutoring scenarios ("get help") are reused in the tele-exam scenario. Our goal is to define and implement components within this tele-teaching framework specialized on other topics as well – and then integrate the different frameworks to a larger whole.

Our current research direction is to implement a generic awareness and monitoring service, which could also be used in tele-exams to pass feedback about the progress of the students back to the teacher. In other settings these elements would become more cooperation aware. An example is a shared workspace, where students are collaborating directly and need to know, when an artifact by a group member has been finished.

We chose the tele-teaching domain to validate our frameworks and components. However, the frameworks are designed to solve common problems in cooperative work. It remains to show that they can be reused in other settings by an actual implementation for those areas, as in workflow or distributed on-line help scenarios.

8. 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] R. Aditham, R. Jain, and M. Srinivasan. Interest Based Collaboration Framework. In *Proceedings of the sixth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* [10], pages 75–80.

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern–Oriented Software Architecture – A System of Patterns*. John Wiley & Sons, Inc., 1996.

[3] P. Dourish. *Open Implementation and Flexibility in CSCW Toolkits*. PhD thesis, University College London, June 1996.

[4] S. Downes. *Web-Based Courses: The Assiniboine Model*. Assiniboine Community College, Canada, September 1997. http://www.assiniboinec.mb.ca/user/downes/naweb/am.htm.

[5] D. F. D'Souza and A. C. Wills. *Objects, Components and Frameworks With Uml: The Catalysis Approach*. Object Technology Series. Addison-Wesley, May 1998. not yet published.

[6] K. Franze, O. Neumann, A. Schill, and S. Stoecker. An Infrastructure for Collaborative Teleteaching. In *Proceedings of the sixth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* [10], pages 341–346.

[7] S. R. Hiltz. Teaching in a Virtual Classroom[TM]. In *International Conference on Computer Assisted Instruction ICCAI'95*, March 1995. http://www.njit.edu/njIT/Department/CCCC/VC/Papers/Teaching.html.

[8] J. Hummes, A. Karsenty, and B. Merialdo. Active Annotations of Web Pages. In R. Alton-Scheidl, R. Schmutzer, P. P. Sint, and G. Tscherteu, editors, *Voting, Rating, Annotation – Web4Groups and other projects: approaches and first experiences*, volume 104 of *Schriftenreihe der Österreichischen Computer Gesellschaft*. Oldenbourg Verlag, Wien, München, 1997.

[9] J. Hummes, A. Kohrs, and B. Merialdo. Software components for cooperation: A solution for the "get help" problem. In *COOP'98: Third International Conference on the Design of Cooperative Systems*, Cannes, France, May 1998.

[10] IEEE. *Proceedings of the sixth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Cambridge, MA, June 1997.

[11] D. B. Lange, M. Oshima, G. Karjoth, and K. Kosaka. Aglets: Programming Mobile Agents in Java. In T. Masuda, Y. Masunaga, and M. Tsukamoto, editors, *Worldwide Computing and Its Applications – International Conference Proceedings*, volume 1274 of *Lecture Notes in Computer Science*, pages 29–42, Tsukuba, Japan, March 1997. Springer.

[12] T. W. Malone, K. R. Grant, K.-Y. Lai, R. Rao, and D. Rosenblitt. Semistructured Messages are Suprisingly Useful for Computer–Supported Coordination. In I. Greif, editor, *Computer–Supported Cooperative Work – A Book of Readings*, pages 311–331. Morgan Kaufmann Publishers, 1988.

[13] T. W. Malone, K.-Y. Lai, and C. Fry. Experiments with Oval: A Radically Tailorable Tool for Cooperative Work. *ACM Transactions on Information Systems*, 13(2):175–205, 1995.

[14] M. Mühlhäuser, editor. *Cooperative Computer–Aided Authoring and Learning*. Kluwer Academic Publishers, 1994.

[15] M. Mühlhäuser and M. Richatz. Instructional Strategies and Processes. In Mühlhäuser [14], chapter 12.

[16] M. Mühlhäuser and J. Schaper. The Nestor Reference Architecture. In Mühlhäuser [14], chapter 5.

[17] A. Mørch. Three levels of end-user tailoring: Customization, integration, and extension. In M. Kyng and L. Mathiassen, editors, *Computers and Design in Context*, chapter 3, pages 51–76. The MIT Press, Cambridge, MA, 1997.

[18] ObjectSpace Inc., Dallas, Texas. *ObjectSpace Voyager – The Agent ORB for Java – Core Technology User Guide*, July 1997. http://www.objectspace.com/voyager.

[19] O. Sims. *Business Objects – Delivering Cooperative Objects for Client–Server*. IBM McGraw–Hill series. McGraw–Hill, 1994.

[20] Sun Microsystems Inc., JavaSoft, 2550 Gracia Avenue, Mountain View, CA 94043. *Java Beans 1.0 API specification*, October 1996. http://java.sun.com/beans.

[21] M. Turoff. Designing a Virtual Classroom[TM]. In *International Conference on Computer Assisted Instruction ICCAI'95*, March 1995. http://www.njit.edu/njIT/Department/CCCC/VC/Papers/Design.html.

[22] D. Wan and P. M. Johnson. Computer Supported Collaborative Learning Using CLARE: the Approach and Experimental Findings. In *ACM Conference on Computer Supported Collaborative Work*, pages 187–198, North Carolina, October 1994. Chapel Hill. file://ftp.ics.hawaii.edu/pub/tr/ics-tr-93-21.ps.Z.