COINED
Collaborative Object INteraction in EDucation
An environment for distributed object role play

Till Schümmer and Petra Kösters

Cooperative Systems, FernUniversität in Hagen
Universitätsstr. 1, 58084 Hagen, Germany
Till.Schuemmer@fernuni-hagen.de

Abstract. This paper discusses object-first approaches for better understanding the interaction between objects in an object-oriented software system. We analyze different approaches for teaching object interaction and derive the need for tool support that enables small groups to simulate objects in a virtual object-interaction role play. To make this concrete, COINED, a groupware application for object role plays, is presented. It models a role play between students where each student directs a set of objects and thereby reacts to messages sent to these objects. COINED visualizes the interaction between the objects and further supports the discussion of the objet interaction as well as the creation of shared artifacts from interaction logs.

1 Introduction

Understanding and teaching object-oriented concepts is still a challenging task. The students need, among others, to understand what objects and classes are and how objects interact with each others in order to solve a specific problem. These challenges are different to challenges in traditional teaching of procedural software development. While teaching procedural software development focuses on empowering the students to create and implement algorithms, the object-oriented view should first teach how to identify actors and how to describe interaction between these actors. Algorithms can then be understood as interaction between the different objects instead of procedural recipes.

This shift in focus can be compared to the evolution from instructional learning where the teacher acts as an imperative information provider who transfers information to the students to problem-based learning (PBL) [10] where the students interact with each others to solve a problem. In PBL, the teacher creates the context for a problem and challenges the students to explore different solutions for this problem. In this process students contribute their knowledge and conception of reality. Problems in PBL should be closely related to real life problems. In the case of object-oriented software development, such problems can be derived from the daily project business in software development. One important component of PBL is that students interact in a group that is supported by the

teacher when needed. This is one reason, why PBL has gained major interest in the field of computer-supported collaborative learning (CSCL). For the area of software development education, team-based PBL approaches are in addition very close to real project work where software developers also interact in teams to solve a specific problem.

When reviewing the state of the art in teaching object-oriented software development, we unfortunately still observe many cases where traditional instructional teaching methods are used to transfer knowledge about the craft of programming to the students. Consequently, these approaches often teach object-oriented software development as if it was procedural software development and as a result, the students often approach object-oriented software development taking an algorithmic and procedural perspective. Although the students are often able to write nice code inside a method, the interplay of the different objects is to our experience in most cases not well understood.

Joe Bergin shared this observation in a post of a objects-first versus objects-late discussion at SIGCSE [4] where he nicely outlined how a new pedagogy should look like:

> If you want to teach a new paradigm [...] then you need to use new pedagogy. [...] You can't teach OO with a piece of chalk. You need to scaffold the process. [...] This is because there are no 3 line OO programs that are interesting. [...] Don't try to teach just through code. Look at the Role Play papers in recent SIGCEE proceedings and at the OOPSLA Educator's symposium reports (or on my web site). Use metaphor. Use games. Use cool ideas/demos for OO the way you know to use cool ideas for other things. [...]
> Learn the craft yourself (maybe you have). In Java prefer interfaces to inheritance. Use simple design patterns that you can motivate outside the coding world (Strategy, decorator, ...). Prefer small classes to large. Prefer many (simple) objects to few (complex) objects. Put complexity in the interactions, not the objects. [...] Students need to look at a big enough program (either in code, or in metaphor, or in role play...) that the interactions between simple things becomes apparent.[1]

We fully agree with this quote and want to repeat that the interaction between objects is to our experience one of the most important aspects when teaching object-oriented software development. But how can we make the interaction between objects clear to students? And how can students learn to design interaction between objects in a PBL setting? How can such an approach be supported as a group exercise that makes sense for the students also in the context of larger software development projects?

These questions were our challenges for writing this paper. We will first summarize current approaches for teaching object interaction (chapter 2). Our focus will be on role play approaches. Although role play approaches are from our perspective the right way to go, we identified problems with these approaches especially in the context of distance education, in situations where traceability

[1] http://listserv.acm.org/scripts/wa.exe?A2=ind0403D&L=SIGCSE-MEMBERS&P=R2575&I=-3

is needed, or with respect to the balance between efforts and benefits. In section 3, we will present new tool support for collaborative object role play interaction, the COINED environment.

2 Teaching Object Interaction

Several proposals have emphasized the importance of object interaction in teaching object-oriented software development. Díaz et al. [6] have highlighted that message passing is one of the fundamental concepts that has to be learned by students. We emphasize this observation and propose that students should understand that the process of sending a message is conceptionally different to the process of invoking a method.

While invoking a method assumes that the invoker can control what the system will do, the metaphor of sending a message is assuming that there will be a receiver who will react to this message in a meaningful way. How this is done is only determined by the receiver.

When looking at tools and processes for supporting an object-first view in teaching and learning, we can observe that this fundamental difference is reflected to a different degree. Basically, we can distinguish between two approaches: use-oriented approaches and simulation-oriented approaches. We will discuss both approaches in the remaining part of this section and identify challenges for a next-generation role play support tool.

1. Use-oriented Approaches. This class of approaches provides the student with a set of simple objects and puts the student in the role of the client. The objects themselves are represented and enacted by a computer. This means that the objects need to be implemented before they can be used. Often, this implementation is performed by the instructor and the student creates code using these objects. In a next stage, the student may also derive new classes from the predefined set of classes and thereby extend the object space.

Examples for supportive environments for use-oriented approaches are Karel J. Robot [2] or Alice [5]. In both systems, the students can interact with objects and see the effects of messages sent to these objects. However, the mental model of this interaction is based on the metaphor of objects that can be controlled from outside. The student becomes the master of a remote control, at least for the first interactions with the objects.

In [5], the authors evaluated their teaching with Alice and concluded that the students could get a good understanding of objects, relations between objects and method invocation. However, we still see some problems with the approaches, especially when considering them from a PBL perspective: firstly, the examples are part of a simulated micro-cosmos that is different from real world problems and secondly, the interaction in such a micro-cosmos is designed as student-micro-cosmos interaction, which is in contradiction to group-based learning.

BlueJ [8] provides a different object-first approach, that is not restricted to visual objects. Instead, any object can be presented as part of an UML diagram.

Users can invoke methods of the objects shown in order to change the state of the object network. While this is a good approach for communicating static object structures and object-oriented modularization, it has problems in teaching the dynamics of an object-oriented software [9].

As Karel J. Robot and Alice, BlueJ only supports single students working with the objects. There is no group support in the tool.

2. Simulation-oriented Approaches have a special emphasis on teaching the dynamics of object interaction. In these approaches, students simulate objects. A very prominent class of simulation-based approaches is based on the the metaphor of a real or virtual role play [3, 1].

In a bootstrap phase, an initial set of objects is created and students are assigned to these objects. They have knowledge of other objects and can send messages to these objects (i.e., to the students representing the object). After receiving a message, the student looks up an appropriate script that tells him how he should react to the message. In [3], the author presents a diagram notation for documenting the interaction in a role play. Each student documents his object by means of a CRC card. This card is placed on a whiteboard. Relations between objects can be drawn on the whiteboard and each message that is sent to an object is documented using an object interaction diagram notation. This is from our perspective one of the biggest deficits of the approach: The manual creation of the diagram which is time consuming and error-prone.

VirPlay [6] transfers the idea of role plays to a virtual multi-user environment. Users are represented as humanoid avatars who simulate their personal object. They can send and react to messages. When users send a message to another user (i.e., when an object sends a message to another object), the VirPlay environments animates this and shows a virtual ball that is thrown from one actor to the other. A spotlight visualizes who is currently playing.

One problem with VirPlay is that it lacks a visualization of the message history. Students cannot easily see which messages have been sent before. Another problem is that the relations between the objects are not visible.

Challenges. The state of the art shows that there is no silver bullet for teaching all aspects of object-orientation. But when looking at the experience made with the different approaches and combining them with our own approaches in teaching object-orientation, we arrive at five principles and challenges that should be considered when teaching the dynamics of object-oriented software:

1. *Object interaction* first.
2. *Messages* are more important than methods.
3. *Objects* are more important than classes.
4. Objects know their *interaction partners.*
5. Students need an *overview* while respecting *encapsulation.*

The discussion of the role play approaches further showed that the interaction of objects can be well communicated to the students by involving them in a role play. Considering this and the potentials of PBL, we can state two additional principle:

6. *Group interaction*: Students should collaboratively explore object interaction and discuss alternatives.
7. Group interaction can be supported by a *groupware system*.

The latter point is especially important when teaching distributed students. However, it also helps to activate students, especially when they can perform parallel actions (e.g., discuss and simulate the object interaction at the same time).

Finally, the motivation for performing the role play needs to be clear to the students:

8. *Reflection*: the object interaction should be tracked and visualized to support the reflection process.
9. *Code generation*: If the system provides means for creating real artifacts from the interaction, the students can see a tangible result that further motivates them to perform the role play.

To approach these challenges, we developed COINED, which will be presented in the next section.

3 COINED

COINED is a collaborative application that supports virtual co-located or distributed role play simulations of object-oriented systems (addressing issues 6 and 7 of the previous section). Developed as a part of the Eclipse IDE, it is intended to be used by a group of students in introductional courses as well as in advanced courses. In the first case, the students will be provided with a set of pre-fabricated objects and their task is to simulate the interaction between these objects. In the latter case, the students start with a blank object interaction diagram and add objects as well as methods on the fly while simulating interaction that is required to address the requirements of a feature of a system that these students are asked to build or extend.

We will show how the system works by describing the support for the principles and challenges of the previous chapter.

3.1 Object interaction first

The core idea of COINED is that the students simulate the interaction in an object-oriented system. When the users enter a project, they see an object diagram containing all instances that currently exist in the simulated program (cf. Figure 1–right).

The diagram in addition shows the interaction between the objects. A message trace provides an alternative view of the dynamics in the system. Messages that await a response are shown with a different color in the message trace and in the diagram. To keep the diagram easy to read, students can control the number of steps in the history that are shown in the diagram. All this contributes to an object interaction based understanding of the simulated system (principle 1).

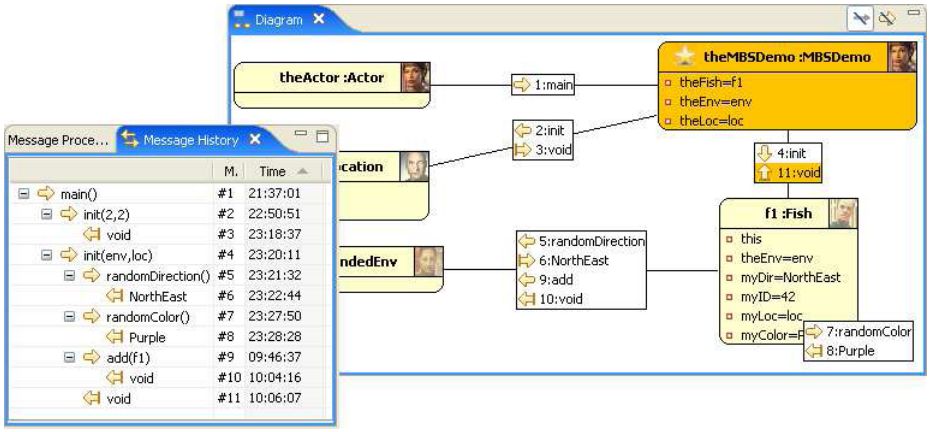

Fig. 1. Message history and interaction diagram in COINED

3.2 Messages are more important than methods

Each student can take responsibility for objects by becoming their *director*. COINED will prompt the director of an object for actions whenever the directed object becomes active (i.e., when it receives a message). To simulate the application, the director of an active object can send a message to another object. This will make the receiving object active and its director will be able to look at the message and perform changes (including the possibility of sending new messages to other objects).

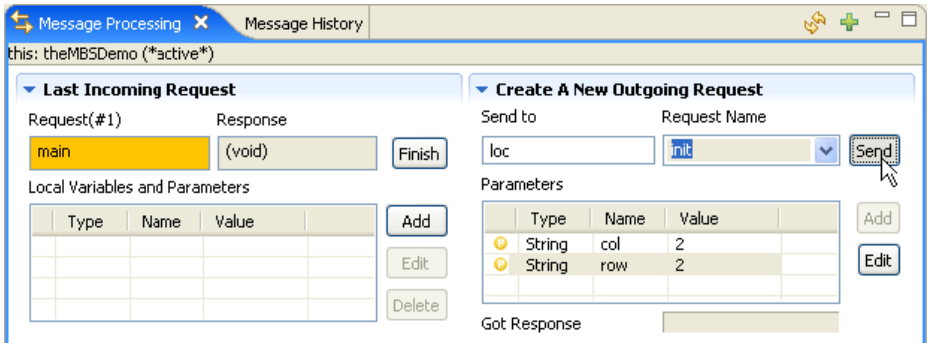

Fig. 2. Message processing in COINED

Messages are processed in the message processing dialogue shown in Figure 2. While the diagram of Figure 1 is shared among all participants, the message

processing dialogue can only be seen and manipulated by the director of the active object.

A director can respond to a message either with a reply message or with a *"does not understand"*-signal (note that this is the message handling behavior found in Smalltalk).

Whenever a director generated a reply message, the object remembers that it has a method to invoke when a message with the same signature arrives. This means that methods are derived from a successful message handling (principle 2). When such a message arrives again, the system shows the director how a comparable message was processed in the past. In addition, these message signatures are added to the object's interface.

3.3 Objects are more important than classes

Directors can agree to make similar objects instances of the same class. COINED supports the required coordination by providing an embedded chat in Eclipse. In this case, the interfaces of the objects are merged. A second reason for creating a class is that the user feels the need to create a second instance that is comparable to an existing first instance.

Note that objects "without" a class are connected to an implicit class. This means that students will start with a blank object and extend this object when needed. However, they are implicitly acting on the class that was created for the new object.

Allowing classless objects is especially important when students start to simulate an object space in an early stage of the design. By interacting with classless objects, they can express their ideas of what the object can do and understand what others expect the object to do. This is an important learning process that in a next step enables them to create real classes for such objects (principle 3).

As it was the case for simple objects, students can become directors of the class and control all instances of the class. Interfaces are connected to the class and message processing histories are aggregated for all instances of the class.

3.4 Objects know their interaction partners

Objects can only send messages to objects that they know (principle 4).The system enforces this by allowing only local fields' contents as message receivers. The only exception for this rule is the delivery of an initial message: To start a role play episode, any user can select an object and send an initial message to this object. However, since this message comes from "nowhere", there is no way to store the result of this message in another object.

Since parameters, local attributes, and instance attributes are all together presented as potential message receivers, it is always clear which objects are known in the current context.

3.5 Encapsulation and overview

Restricting the interaction partners to those objects that are known in the current context also helps to better understand the concept of encapsulation. Only the director of an object can use and manipulate the content of a local attribute. Encapsulation is enforced by this local view of the system.

The local view, however, poses a problem to the goal of providing an understanding of the interaction in the whole system. To reach this goal, students should be allowed to see all objects, their relations, and theirs state in order to follow the messages as they travel through the object network.

In COINED, we decided to provide the global view in the object interaction diagram but enforce encapsulation in the interaction with the shown objects (challenge 5 of the previous section). Since this is only a view, the students will not be able to make use of the shown objects for processing a message. But the global view of objects can provide hints on which objects need to be related and used for processing a specific message.

3.6 Incentives: Code generation.

The goals mentioned in the previous sections lead to an interaction process between the different directors that poses an additional overhead on them. The restriction that an object needs to know an other object before it can send a message to it is slowing down the message processing. One could argue that the director (the human user) can see the full interaction diagram and without this restriction call any object that he sees on the screen.

However, it is important to communicate that the COINED system has the goal of substituting the computer as execution engine and thus keeps the human user as close as possible to the computer as an execution engine. The benefit from doing so is that the interaction logs can be used for two purposes: they help the students to reflect on the interaction that they performed (challenge 8 in the previous chapter) and the provide the required data to generate code and automate the message processing. The latter is the first step for transforming the simulation into a real and running piece of software (challenge 9 of the previous chapter).

COINED currently creates classes with all used methods and attributes. Although this is not yet sufficient for capturing the object interaction in code, it is a first step that can reduce the work required for a future implementation of the simulated system.

4 Conclusions and Future Work

It is still a challenge to teach object-oriented software development. Problem-based learning is a promising way for facing this challenge. In this paper, we have investigated several object-first approaches and corresponding tool support

and concluded that most tools do not fully address the domain specific requirements of a full object-first approach. Especially, they often lack in supporting collaboration as it would be needed to follow a PBL approach.

To overcome this deficit, we proposed the COINED system, a groupware application that supports a group of students in a co-located or distributed object role play. As all groupware systems, it combines a set of "intentional group processes plus software to support them" [7]. The group process models a role play between students where each student directs a set of objects with the goal of react to messages sent to these objects. The supportive software visualizes the interaction between the objects and further supports the discussion of the object interaction and the creation of shared artifacts from interaction logs.

We are currently working on making the system available under an open-source licence.[2] First experiences showed that the system can be used to better understand the roles of objects, especially in the early phases of design. However, these experiences are up to now based on personal interaction that the authors of this paper had with other students. We have not yet performed a formal study on the effects of using COINED in regular courses. Such a formal evaluation is planned as future work.

In addition, we ar currently investigating to what extent user interface simulation can be supported with a comparable role play approach. The vision for an extended system is that the students can simulate a full software system using both user interface elements and internal objects.

With respect to code generation, this will face us with an additional challenge. We will not only need to create a more complete code factory for the internal domain object interaction but also connect this code to sketched user interfaces.

References

1. S. K. Andrianoff and D. B. Levine. Role playing in an object-oriented world. In *SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 121–125, New York, NY, USA, 2002. ACM Press.
2. J. Bergin, M. Stehlik, J. Roberts, and R. Pattis. *Karel J. Robot A Gentle Introduction to the Art of Object-Oriented Programming in Java.* Dream Songs Press, 2005.
3. J. Börstler. Improving crc-card role-play with role-play diagrams. In *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 356–364, New York, NY, USA, 2005. ACM Press.
4. K. B. Bruce. Controversy on how to teach cs 1: a discussion on the sigcse-members mailing list. In *ITiCSE-WGR '04: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 29–34, New York, NY, USA, 2004. ACM Press.
5. S. Cooper, W. Dann, and R. Pausch. Teaching objectsfirst in introductory computer science. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, page 191195. ACM Press, 2003.

[2] See http://www.pi6.fernuni-hagen.de/en/projekte/coined.html.

6. G. Jiménez-Díaz, M. Gómez-Albarrán, M. A. Gómez-Martin, and P. A. González-Calero. Software behaviour understanding supported by dynamic visualization and role-play. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 54–58, New York, NY, USA, 2005. ACM Press.
7. P. Johnson-Lenz and T. Johnson-Lenz. Consider the groupware: Design and group process impacts on communication in the electronic medium. In S. Hiltz and E. Kerr, editors, *Studies of Computer-Mediated Communications Systems: A Synthesis of the Findings*, volume 16. Computerized Conferencing and Communications Center, New Jersey Institute of Technology, Newark, New Jersey, 1981.
8. M. Klling, B. Quig, A. Patterson, and J. Rosenberg. The bluej system and its pedagogy. *Computer Science Education*, 13(4):249–268, 2003.
9. N. Ragonis and M. Ben-Ari. On understanding the statics and dynamics of object-oriented programs. *SIGCSE Bull.*, 37(1):226–230, 2005.
10. D. Woods. *Problem Based Learning: how to gain the most from PBL*. McMaster University Bookstore, Hamilton, ON, Canada, 1994.