Software Behaviour Understanding Supported by Dynamic Visualization and Role-Play

Guillermo Jiménez-Díaz, Mercedes Gómez-Albarrán,
Marco A. Gómez-Martín and Pedro A. González-Calero
Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid
Profesor José García Santesmases, s/n. 28040, Madrid, Spain
+34-913947646

gjimenez@fdi.ucm.es, {albarran, marcoa, pedro}@sip.ucm.es

ABSTRACT
Visualization techniques are commonly used in computer science, particularly for understanding the interactions intrinsic in the object-oriented paradigm. The visualization effectiveness improves if the student takes an active role during the learning process. In this paper we propose an active learning approach that lies in using role-play simulations in a virtual 3D environment.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Object-Oriented Issues, Graphics-Visualization, CS1/2, Pedagogy.*

General Terms
Design, Human Factors.

Keywords
Software comprehension, dynamic visualization, role-play, scenario diagrams.

1. INTRODUCTION
Understanding software behaviour is a key aspect of software maintenance and reuse. However, the task of software understanding is difficult, especially when working on object-oriented systems, where the objects that make up the system interact among themselves to get the expected behaviour. Sometimes we need to understand which interactions occur when an event happens or when an object triggers a specific message. For example, before reusing an object-oriented framework, a developer needs to understand how the instances of framework classes collaborate at runtime and how they will interact with the objects created by extending the framework in order to develop a new application.

ITiCSE'05, June 27–29, 2005, Monte de Caparica, Portugal.

Understanding these interactions in complex systems can become a very hard task. On the one hand, reading documentation about the classes and their architecture does not ease this task much. On the other hand, tracing the software execution in a development environment can be even more difficult than following the documentation approach. Commonly, the user gets lost in a mess of message passing and leaves the task.

Several software visualization approaches have focused on illustrating the execution of object-oriented applications and their architecture. However, most of them cannot be used for education because they do not take into account that a student needs some kind of motivation to successfully understand the interactions between objects in these kind of systems.

In this paper we describe an approach for understanding the interactions among objects using role-playing in a 3D environment with an anthropomorphic metaphor of the object-oriented paradigm. The student can play the role of an object with other avatars (the representations of the user and the objects in the 3D environment), like in classroom role-play. Moreover, the student can interact with them in order to obtain more information about the class of the corresponding objects and the messages passed during the simulation.

The organization of this document is as follows. Section 2 describes the foundations of the approach, that is, role-play and dynamic visualization. In this section we have also included a summary of related work. The details of the approach are described in Section 3, including the visual metaphor we have employed, the interactivity between the user and the simulation, and a sample use case. Finally, we conclude with the future work.

2. BACKGROUND KNOWLEDGE
This section briefly presents role-play and dynamic visualization, the two aspects our approach combines. A summary of related work is also included.

2.1 Role-play
Role-play is a kind of active learning where students learn complex concepts –hard to understand by means of abstract explanations– while they simulate a scenario [2]. The role-play supervisor provides information about the characters in the play to each participant and, depending on the degree of freedom given to the actors, a predefined script of the scenario that they are going to perform. During the role-play, participants will interact,

learning from themselves, the other participants and the played roles [3].

A beneficial effect of role-play is that students not only learn about the played roles but also they have the opportunity to see what they understand. If the student understands the roles of a scenario, then she will know how and with whom to interact [2].

Role-play is typically employed to instruct in social skills but several authors have suggested using it to teach topics from computer science such as hardware design, formal methods, backtracking or object-oriented design [1]. The work in [2] points out that humans are a good metaphor for objects in the object-oriented paradigm, because "people are autonomous, encapsulated actors, who interact with other similar objects". For this reason, when a student takes part in a role-play that represents the execution of an object-oriented application she is "putting himself/herself into an object's shoes".

Despite the importance of role-play in teaching computer science topics –it was included in the list of pedagogical patterns [2]– its use in instructional software for object-oriented programming has not been reported. In contrast, there are several collaborative web-based environments such as TimeScope [13] and Fablusi [7] to teach history using role-playing. In these environments, the participants interact among themselves playing a proposed scenario in a forum or chat. All the participants and the supervisor are humans because the scenarios are not scripted role-playing and the use of intelligent agents is very difficult.

2.2 Dynamic visualization

The adage 'a picture is worth a thousand words' accurately reflects the enormous communication ability that images provide. Human beings are good at processing visual information. These two facts, together with the rise of modern human-computer interfaces, have led to the massive use of software visualization as a pedagogical tool in the task of teaching programming and as a key piece in software comprehension. Software visualization is the use of typography, graphic design, computer animation, cinematography with graphics technology to facilitate the understanding and effective use of software [6]. Most of the software visualization approaches focus on illustrating algorithms, program flow and information. However, we are interested in approaches to visualize both static architecture and behaviour of software systems. The process of modelling the latter is called dynamic visualization.

Dynamic visualization is commonly used in reverse engineering and software comprehension because it makes easier to understand the complex interactions. Describing a dynamic process using words is more troublesome than viewing it at work. Besides, in a visualization it is easier to understand that some events are happening at the same time.

In software visualization, UML models are the most common technique to visualize the architecture and behaviour of an object-oriented system. The class and deployment diagrams provide an architectural view of the system, while collaboration, state charts and activity diagrams are used to represent the dynamic behaviour.

Several visualization approaches extend the use of UML diagrams. Grundy *et al.* [6] propose a complex model that combines UML models with its own visual language for static and dynamic visualization of software architecture.

Other software visualization tools employ metaphors to represent the software architecture. Metaphors provide a way of transmitting software information in a representation that can be easier to understand. A commonly used metaphor is 3D virtual microworlds where the classes are represented by platforms with incrusted elements –like columns and spheres– to represent the class methods and attributes. The user can navigate through this microworlds in virtual reality [9, 15].

A more complex metaphor is used in 3D City [11], where countries, cities and buildings are employed in representing the different elements of the software. This kind of approach is mainly oriented to represent the static architecture of the software and colours and sizes are used to represent different software metrics. 3D City also contains cars that drive from one building to another to represent the dynamic behaviour of the system. But they are only used to represent metrics about the amount of message passing and not for understanding interactions.

The main disadvantage of most of these tools is that they were not developed thinking about an educational approach because their motivation was debugging, software metrics visualization or really complex system understanding. In contrast, other approaches were developed for pedagogical purposes. *Thaden et al.* [14] proposes the animation of UML diagrams to visualize the execution of arbitrary Java programs. The classes and objects that appear in an UML diagram are translated into animated 3D blocks.

Alice [4] is another interesting work because of the way it shows the behaviour of an object-oriented program. Alice is a development environment that uses an anthropomorphic metaphor to represent objects. The student designs a world and populates it with characters, whose behaviour is also implemented by the student. When the student runs the developed program, she can visualize the behaviour and the state of the characters in the world. The main disadvantage of Alice is that it cannot be used for the dynamic visualization of arbitrary object-oriented programs.

3. OUR APPROACH

We propose to build 3D worlds automatically so that our Computer Science students can understand the software behaviour when an initial message has occurred in an object-oriented system. Instead of building the whole message passing sequence when the application runs its main method, the simulation will perform an execution scenario. In this way, the learning task of understanding the dynamic behaviour of the application will be divided into several pieces and the student can focus on one concrete issue at a time.

We have selected scenario diagrams to represent the scenarios that the instructor considers useful for understanding a concrete interaction. A scenario diagram is a notation for visualizing the message flow in object-oriented systems. Commonly, UML sequence diagrams are employed to characterize this notation.

As in [14], we are using the XMI representation of UML diagrams in order to create our visualization. However, we do not translate it into a language like VRML. Instead, we transform it into a script that contains information about the objects in the scene and the messages that they will pass during the role-play.

This script also contains the design information that can be useful during the simulation.

The simulation of the role-play will be performed in an interactive 3D metaphorical world. The student will have an avatar, which initiates the role-playing and can interact with other avatars during the simulation. The student's avatar is in the center of the scenario we have developed and the avatars of the different objects are situated around it. Since message passing grows exponentially with the number of participants, we limit the number of avatars that can appear in a scene (seven avatars at most, including the student's avatar). When the student starts the role-play with the first message, the avatars interact among themselves, by passing the messages from the performed scenario diagram. The student's avatar can resume the role-playing at any time and ask the other avatars in order to extract design information. As an alternative, the student could also adopt the role of one of the objects in the scene, trying to figure out and understand the behaviour of the object during the current interaction.

3.1 The metaphor

We are going to represent the following sequence diagram elements within the metaphor:

- Actor: The actor is the entity that starts the interaction. In this metaphor the actor will be the student's avatar.
- Object: The objects are the rest of characters that take part in the role-playing. The character shows the object name and the object class. In the metaphor, the active object holds a ball and a spotlight illuminates it.
- Message: When object A passes a message to object B, the avatar of the object A says the name of the message addressee, the text of the message and throws the ball to the avatar of the object B. The spotlight focus will change from avatar A to avatar B.
- Self-message: An object can pass a message to itself. When this happens, the avatar throws the ball upwards, saying its own name and the text of the message. The focus stays on the avatar.
- Return arrow: When object B has concluded the execution of a message passed by object A, the focus returns to object A. If avatar B throws the ball to avatar A, the student could suppose incorrectly that an event occurs. Instead of that, avatar B will roll the ball until it reaches avatar A and says the message "*MessageName is done resulting ReturnObject*" –if this message has returned something– or "*MessageName is done*" –if the message has not returned anything. Finally, the spotlight will illuminate A.
- Object creation: For creating a new object we need to represent the class of that object. Instead of a character, a class will be a block with the class name. When an avatar wants to create an object, it will say the message *"ObjectName, construct yourself as a ClassName object with <parameters>"* and it will throw the ball to the block. This message will cause the object avatar to appear with the ball and the focus.

In addition, we want to illustrate the state of an object and its references to other objects. As in a graphic adventure, every avatar has an inventory, which contains several items representing the value of its attributes, a reference list of its collaborators and a set with the distinguished local variables in the current scope. The inventory will be very useful for the student when she performs the role of an object, because it also contains information about the object design. We propose the use of Javadoc-like documentation and explained fragments of source code.

Finally, the ball does not only represent the control flow but also contains the information about the last message passed. With a zoom-in, the user can "enter" in the ball and explore the message signature and parameters and the return value.

3.2 Interactive simulation

The main pedagogical benefits obtained using role-play are because it is an active approach for learning. We agree with the thesis of "visualization technology is of little educational value unless it engages learners in an active learning activity" [10]. For this reason, we do not want to reduce our approach to show a movie about the interactions among objects. Instead, we want to involve students in the simulation. For this reason, the student's avatar can interact with the remaining avatars in two ways:

- Asking for information. The role-play is a good situation for the student to learn about the design of the classes behind the objects' avatars. The student has timing control of the simulation. She can pause it and ask for information from the other avatars. The questions the user can ask are: *show me the public interface* of your class, *show me your attributes*, *which classes are your superclasses* or *show me the source code that passes this message*. The student can also know the state of the objects in the simulation by consulting their inventories.
- Performing a role. A good way for the student to verify her own knowledge about the system behaviour is to substitute for an avatar and interpret its role. The student can stop the simulation and suggest to an avatar "*Object, I can take care of you*". Then, the student should reply to messages in the expected way. To do that, the student needs some help information like the CRC cards in [3] or the script that encloses the case study described below. The student should pass the messages using actions from graphic-adventure games like *Send <Message> to <Object> using <Parameters>*. If the student makes a mistake during the message passing –a wrong message or any wrong parameter– she will be informed with an error message.

3.3 An example visualization

To illustrate our work we consider the Marine Biology Case study (MBCS), a role-playing exercise referred in [1] and used in the Advanced Placement Computer Science curriculum (http://apcentral.collegeboard.com) to expose students to the idea of intercommunicating objects. We have created a simplified scenario to understand how the main class creates a fish. The sequence diagram corresponding to this scenario is shown in Figure 1.

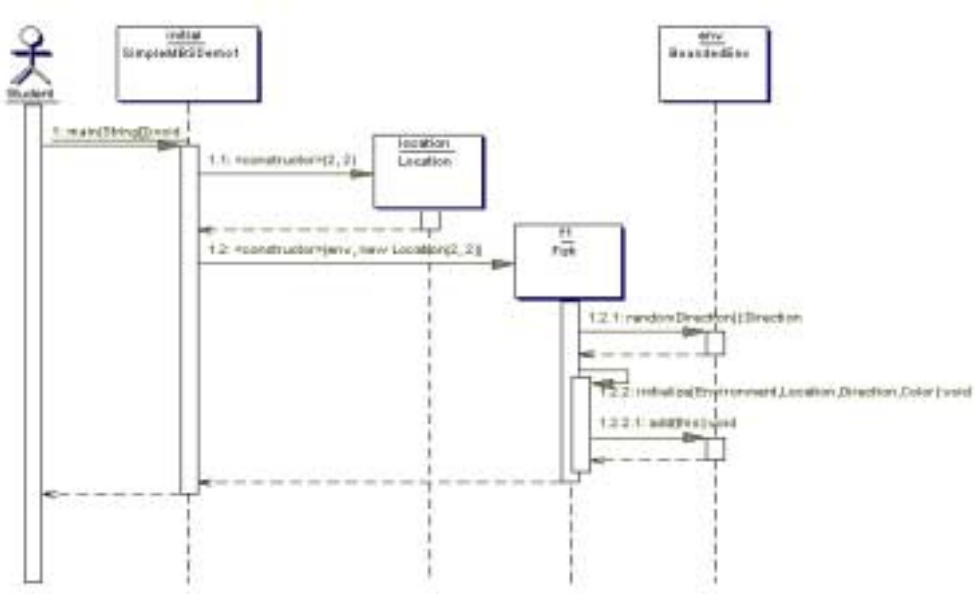

Figure 1. Sample scenario extracted from MBSC

Using the Java MB role-play script, we have created the following role-playing simulation. The student's avatar appears in the middle of the scene and, around it, two avatars, <*initial:SimpleMBSDemo1*> and <*env:BoundedEnv*>, and two blocks, which represent the classes Location and Fish. The focus is on the student's avatar. This one starts the simulation throwing a ball to *initial* and saying "*initial, do main with args*". When the ball reaches *initial*, the focus is on it. Then it looks for the *Location block* and throws the ball saying "*location, construct yourself as a Location object with 2 and 2*". When the ball touches the block, an avatar called <*location:Location*> appears. Then this avatar says, "*Constructor is done*" and the ball returns rolling to *initial*. Now, *initial* throws the ball to the other block and says "*f1, construct yourself as a Fish object with env and location*". As a result, the avatar <*f1:Fish*> appears from the block and the focus remains on it. Avatar *f1* looks for *env* and throws the ball saying "*env, do randomDirection*". The ball and the focus reach *env*. Afterwards, it rolls the ball towards *f1* and says, "*randomDirection is done returning direction*". Now the focus returns to *f1* and the avatar throws the ball upwards and says "*f1, do initialize with env, location, direction and color*". After that, it throws again the ball to *env*, but this time it says "*env, do add with myself*" (instead of "with f1" or "with this"). Avatar *env* catches the ball and the focus and returns the ball to f1, rolling it and saying "*add is done*". Now *f1* rolls the ball towards *initial* and says "*constructor done*". Finally, when the ball and the focus are on *initial*, it rolls the ball towards the student's avatar and says, "*main is done*".

Let us suppose that the student says "*f1, I can take care of you*" when *f1* has the ball and after executing the message *env:randomDirection*. Now the student should resolve the message passing of *f1*. In order to obtain help, the student can consult *f1*'s inventory. It contains links to a *Fish* class Javadoc-like document and its source code, an item called *myId* (the fish identifier, represented by an integer), and a list with the following references: *theEnv* (the object representing the environment where the fish is located), *myLoc* (representing its position in the environment) *myDir* (the swimming direction), *myColor* (the fish color) and *this* (itself reference). Besides, the list also contains the references *env*, *loc*, *aColor* and *aDir* (generated by the last message), the local variables at this point. After consulting the information the student is ready to execute the next message passing in the simulation by means of the action *send*. Using an alternative interface, the student should select the correct sender – the reference *this*–, message –*initialize*–, and parameters –the references *env*, *loc*, *aColor* and *aDir*. The avatar will generate and say the corresponding phrase –"*f1, do initialize with env, location, direction and color*". It will throw the ball upwards and the state will be updated, ready to the next message passing.

In Figure 2 we can see the visual aspect of the environment during the simulation described above.

Figure 2. A screen shot from the role-play environment

4. CONCLUSIONS AND FUTURE WORK

In this paper we have described an approach that mixes active learning with visualization techniques to understand the interactions occurring during the execution of object-oriented programs. We have adopted the role-play technique, frequently used to teach object-oriented programming, for involving the student in the learning process. A metaphorical 3D environment is employed to visualize the scenarios of interactions.

We are integrating this approach with a 3D game/visualization engine called Nebula Device [12] because it allows scripting using Tcl, Lua and Python. We are working on deciding which is the most useful way for providing the information extracted from the other avatars and the one used for playing the role of an object. Besides, we need to extend this metaphor, to include the distinctions between object references and primitive data types (integers, booleans…).

Further, we want to extend the student interaction. Currently, the state of an object performed by the student is updated automatically. It would be useful that the student should annotate in the inventory the changes in the current state of the performed object. In addition to that, we want to replace the *send* action interface with a dialogue between the avatars and add a metaphor for the student mistakes during the interaction.

The described approach is being developed using our experience in the system Javy [5], a 3D virtual environment where the student can learn the Java Virtual Machine (JVM) structure and the Java language compilation. Besides, our approach is currently being used in a prototype tool we are developing for understanding object-oriented frameworks [8].

5. ACKNOWLEDGMENTS

This work is supported by the Spanish Committee of Science & Technology (TIC2002-01961).

6. REFERENCES

[1] Andrianoff, S.K., Levine, D.B. Role Playing in an Object-Oriented World. In Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education (Cincinnati, Kentucky, February 27 - March 03, 2002), ACM Press, 2002, 121-125.

[2] Bergin, J., Eckstein, J., Wallingford, E., Manns, M.L. Patterns for Gaining Different Perspectives. In Proceedings of the 8th Conference on Pattern Languages of Programs (Monticello, Illinois, USA, September 11-15, 2001), 2001.

[3] Börstler, J. Object-Oriented Analysis and Design Through Scenario Role-Play. Technical Report UMINF 04.04, Department of Computing Science, Umeå University, Umeå, Sweden, 2004.

[4] Dann, W., Dragon, T., Cooper, S., Dietzler, K., Ryan, K., Pausch, R. Objects: Visualization of Behavior and State. In Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education (Thessaloniki, Greece, June 30-July 2, 2003), ACM Press, 2003, 84-88.

[5] Gómez-Martín, P.P., Gómez-Martín, M.A., González-Calero, P.A. Javy: Virtual Environment for Case-Based Teaching of Java Virtual Machine. In Proceedings of the 7th International Conference on Knowledge-Based Intelligent Information & Engineering Systems (University of Oxford, United Kingdom, 3-5 September), Springer Verlag, 2003, 906-913.

[6] Grundy, J., Hosking, J. High-level Static and Dynamic Visualisation of Software Architectures. In Proceedings of the 2000 IEEE International Symposium on Visual Languages (Seattle, Washington, USA, September 10 - 13, 2000), IEEE Computer Society, 2000, 5-12.

[7] Ip, A., Linser, R., Naidu, S. Simulated Worlds: Rapid Generation of Web-Based Role-Play. In Proceedings of the 7th Australasian World Wide Web Conference (Coffs Harbour, Australia, 21-25 April 2001), 2001.

[8] Jiménez Díaz, G., Gómez Albarrán, M., González Calero, P.A.: UnderFrame: Understanding Object-Oriented Frameworks Using a Case-Based Teaching Approach, in Poster at 18th European Conference on Object-Oriented Programming. Oslo, Norway, 2004.

[9] Maletic, J.I., Leigh, J., Marcus, A., Dunlap, G. Visualizing Object-Oriented Software in Virtual Reality. In Proceedings of the 9th International Workshop on Program Comprehension (Toronto, Canada, May 12 - 13, 2001), IEEE Computer Society, 2001, 26-38.

[10] Naps, T.L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., Velázquez-Iturbide, J.A. Exploring the Role of Visualization and Engagement in Computer Science Education. In: M.E. Caspersen, D. Joyce, D. Goelman, I. Utting (eds.): Working group reports from ITiCSE on Innovation and Technology in computer science education, ACM Press, Århus, Denmark, (2002) 131-152.

[11] Panas, T., Berrigan, R., Grundy, J. A 3D Metaphor for Software Production Visualization. In Proceedings of the 7th International Conference on Information Visualization (London, England, July 16 - 18, 2003), IEEE Computer Society, 2003, 314-319.

[12] Nebula Device from Radon Labs GmbH Game Development (2004): http://www.radonlabs.de/.

[13] Sharf, J., Hsu, T., Ryan, E. TimeScope: an Educational 3D Multi-User Role-Playing Environment. In Proceedings of the 9th Annual Conference of the Internet Society (San Jose, CA, USA, June 22-25, 1999), 1999.

[14] Thaden, U., Steimann, F. Animated UML as a 3D-illustration for Teaching OOP. In Proceedings of the 7th Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts in 17th European Conference on Object-Oriented Programming (Darmstadt, Germany, July 21-25, 2003), Springer, 2003.

[15] Young, P., Munro, M. Visualising Software in Virtual Reality. In Proceedings of the 6th International Workshop on Program Comprehension (Ischia, Italy, June 24 - 26, 1998), IEEE Computer Society, 1998, 19-26.