Proc. Advanced Visual Interfaces (AVI'2000)
Laura Tarantino (Ed.), ACM Press (2000)

On Better Understanding UML Diagrams through Interactive Three-Dimensional Visualization and Animation

Oliver Radfelder
Univ. of Bremen, Computer Science Dept.
PO Box 330440, D-28534 Bremen, Germany

Martin Gogolla
Univ. of Bremen, Computer Science Dept.
PO Box 330440, D-28534 Bremen, Germany

ABSTRACT

Different approaches support the construction of software by representing certain aspects of a system graphically. Recently, the UML has become common to provide software designers with tools, in which they can create visual representations of software interactively. But the UML is intended to be drawn on two-dimensional surfaces. Our approach extends UML into a third and fourth dimension in a way that we can place both static and dynamic aspects in one single view. By this, we can show behavior in the context of structural aspects, instead of drawing different diagrams for each aspect with only loose relation to each other. We also use the third dimension to emphasize important things and to place less interesting things in the background. Thereby, we direct the viewer's attention to the important things in the foreground. Currently, UML shows dynamic behavior by diagrams which do not change and are therefore static in nature. In sequence diagrams, for example, time elapses from the top of the diagram to the bottom. We point out that behavior is better visualized by animated diagrams where message symbols move from the sender object to the receiver object. Our approach supports the creation of a system as well as the communication of its dynamic processes especially to customers.

Keywords: Visual Interface Design, Visual Interaction, Visualization, Animated Interfaces

1. INTRODUCTION

"UML notation is intended to be drawn on two-dimensional surfaces. Some shapes are two-dimensional projections of three-dimensional shapes (such as cubes), but they are still rendered as icons on a two-dimensional surface. In the near future, true three-dimensional layout and navigation may be possible on desktop machines; however, it is not currently practical." [5]

Three-dimensional information visualization is a fast evolving field in Computer Science. Due to increasing availability of inexpensive hardware acceleration and the development of programming interfaces such as VRML [7] and Java3d [2], the third dimension for visualization is becoming more and more popular for representing complex scenarios.

Our approach combines the benefits from the *third dimension*, from *interactivity* and from *animation* in the visualization of static aspects and dynamic aspects of systems.

Static aspects (1) To enable the user to walk in a model, we use *interactivity*, so that she can examine single symbols or subsystems. (2) We use the *third dimension* to arrange items according to their current importance. As an example, we place interactively chosen class symbols in front of others when we want to study these classes. (3) We use *animation* to smoothly transition from one diagram to another one Instead of showing two different snapshots.

Dynamic aspects: (1) We also use the *third dimension* to place two different aspects of a system in the same view. We will place *dynamic aspects* in the context of the *static* aspect, for example, we will show an interaction in front of a class diagram. (2) We also use *interactivity* to define interaction. We specify interactions by pointing to a class symbol for creating an object symbol and we create message arrows by pointing to the affected object symbols. (3) Finally, we use *animation* to visualize the flow of messages during an interaction which we have defined interactively. This will be done in particular in the context of the static aspects (i.e. in the context of class diagrams).

Some other techniques to help the modeler focusing on certain aspects are *changing of colors* and *changing transparency* to hide or show information.

Up to now, a lot of work has been done in connection with visualization. Due to space limitations we can relate our work with only some other approaches found in the literature.

In [8] the authors show the superiority of three dimensions by means of an experiment. One can also use the third dimension to create layers of two-dimensional diagrams where the same aspect in different environments is connected by way of orthogonal links [3]. [6] uses animation to show changing states.

The rest of this paper, which continuous research started in [4], is structured as follows. In Sect. 2 we present our three-dimensional UML class diagrams and how they are created. We also discuss the benefits of the third dimension, animation and information hiding for this particular kind of diagram. Section 3 discusses the use of three-dimensional interaction diagrams in context of class diagrams and how they are created and layouted. The previously created interactions are presented in various ways. Two types of *ani-*

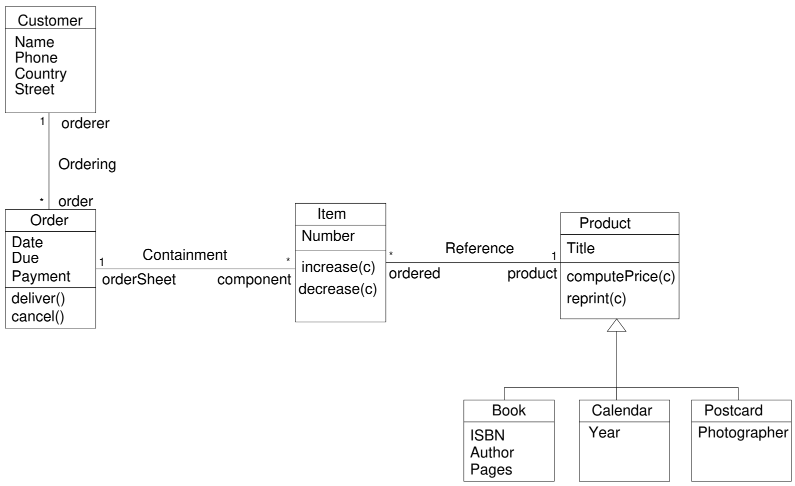

Figure 1: Print Shop Class Diagram

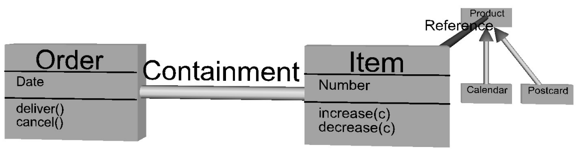

Figure 2: Class Diagram – *Order* and *Item* currently edited

mated interactions are presented in Sect. 4. The paper ends in Sect. 5 with concluding remarks and some thoughts on how to integrate our approach with other software engineering tools.

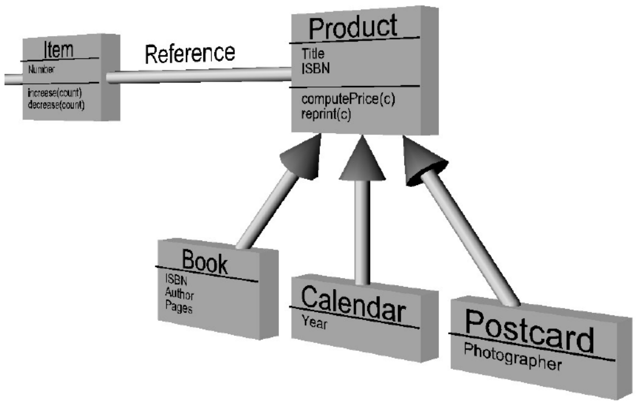

Figure 3: Three-Dimensional Class Diagram

2. CLASS DIAGRAMS

Class diagrams in UML [5] show the static structure of a model without temporal information. Put simply, a class diagram consists of *classes* as rectangles, links between the classes denoting *associations* and special arrows to visualize *inheritance*. *Classes* may be separated into three compartments by horizontal lines. The top compartment holds the name, the middle compartment can show a list of attributes, and the bottom compartment can show a list of operations. Attributes and operations both can be suppressed for clarity. *Associations*, shown as edges between class rectangles, may have a *name* and *multiplicities* and *rolenames* for each end.

Class diagrams can be very complex. If there are many classes, each with a lot of attributes and operations, the diagram loses its main purpose: the visualization of complex systems in a reasonable way. That is the reason why the compartments for attributes and operations may be suppressed in our approach under certain conditions. By suppressing the inner structures of certain classes while showing the structures of others, attention can be directed to a particular aspect of a complex system.

However, sometimes it is not sufficient only to suppress unnecessary adornments. If the aspect we want to visualize concerns two classes on opposite ends of a large class diagram, the class symbols between them may irritate the viewer. We believe it is better to use the third dimension to focus on the classes of interest: We arrange the classes as we would do it in two dimensions and then move the classes, we want to emphasize, to the front *and* let the inner structure appear. At first the user clicks on the classes she is interested in. The selected classes then change their color. When all the classes to examine are selected, the user starts the transformation: The selected classes smoothly move to the front while the others stay in or move to the background.

Although a class diagram usually is read from a file, a user can modify it by adding and deleting classes interactively. New classes can be added in the same way they usually are added in two-dimensional environments: by selecting a menu item and clicking at the point where the class symbol is to appear. Afterwards the symbol can be moved on a virtual two-dimensional canvas to place it properly (for example, under its superclasses and next to associated classes).

Associations and *inheritance arrows* are added similarly, namely by choosing a menu item and clicking on the partic-

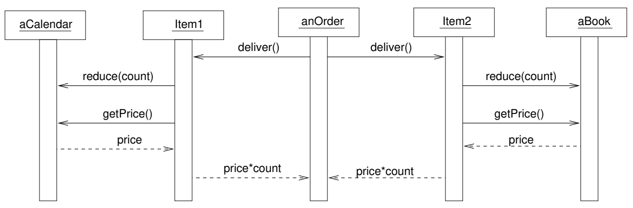

Figure 4: Sequence Diagram – Processing an Order

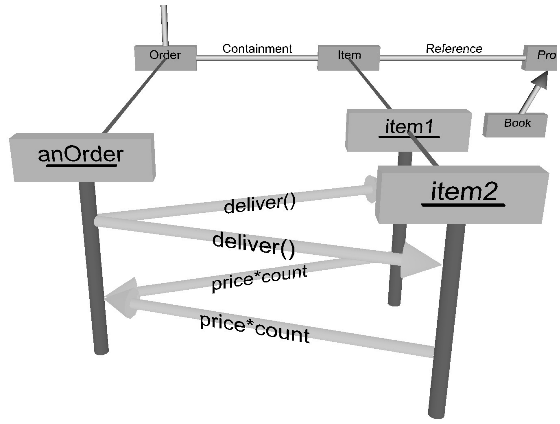

Figure 5: Three-Dimensional Sequence Diagram Processing an Order

ipating classes.

Example: Our example partially specifies structure and interaction within a small print office as it is shown in Fig. 1. The company prints several kinds of *Books*, *Calendars* and *Postcards* which are subclasses of *Product*. A *Customer* may order many different products at once where each ordered product is an *Item*. In contrast to this example, real world models usually are much more complex. Thus, we think that the benefits from the *additional visualization dimension*, *interaction* and *animation* in modeling are even more apparent in larger real world models. For demonstration purposes, however, we use this simple diagram.

Figure 2 is a snapshot of a typical situation in the modeling process: some classes exist already while some others still need to be created. The modeler is currently modifying the class *Order*: the attribute *date* has just been inserted but *due* and *payment* are still missing. When *Order* is complete, the modeler can push the class into the background to create another class. Another option is to select a second class which is somehow related to *Order* and let it move to the front in addition to *Order*. We believe this form of transformation - moving the things of interest smoothly to the front and moving things which lost their particularity into the background - is much closer to the way of perception in the non-virtual reality than hiding the uninteresting parts entirely. In our example we could click, for example, on the *Item* symbol to bring it to the front. In Fig. 3 the entire class diagram is shown, so that attributes and operations are visible.

3. INTERACTION DIAGRAMS

In every interesting system, objects don't just sit idle; they interact with one another by passing messages [1]. UML does not only define how to describe the static structure of a system although this is always the basic building block. In fact, UML consists of more diagram forms concerning dynamic behavior than of those concerning the static structure. Unfortunately, a UML diagram itself is in principle entirely *static*. Usually, behavior may be shown by state symbols, and arrows denote *transitions* between them, as in statecharts. In collaboration diagrams, different phases of execution are separated by numbered method invocations attached to the associations where they occur.

In Fig. 4 we show a sequence diagram where time elapses from the top to the bottom. The diagram shows what typically happens when an *Order* is processed: on each single *Item*, which belongs to this *Order*, the message *deliver* is invoked. The *item* then reduces the number of available pieces, gets the price and returns the lump sum price the customer has to pay. This sort of sequence diagram does not visualize an algorithm but a typical sequence of steps taken. Sequence diagrams are prefect to show different interactions starting concurrently. To be more precise: they are perfect to show *two* interactions starting at the same time. In Fig. 4 *anOrder* sends the message *deliver()* to both *Items* - to the left one and to the right one - at the same time. Although it would be possible to have more than two *items*, it is difficult to draw more than two arrows starting at the same point with no resulting ambiguities. As explained below, this dilemma can be solved in three dimensions.

Describing interactivity by means of two-dimensional graphical tools is an essential element of most CASE tools. Arrows from one object lifeline to another object lifeline are drawn in two dimensions just as we do it in three dimensions: first we click with the mouse on the lifeline of the sender and then we click on the lifeline of the receiver. Thus, we first need to identify the objects in question.

As we mentioned above, we want to describe an interaction in the context of the static structure. Clicking on a class symbol produces an object which immediately moves to its pre-determined place in space. The determined place is calculated by the following rules: each new object is a bit closer to the foreground than the preceding object and each object is in front of the class it is created from. Thus, the x- and the y-coordinates of an object are the same as the coordinates of the class symbol and the z-coordinate is then calculated. This way, the object symbols are placed automatically with no need of user interference and the resulting layout prevents most crossings of lifelines and message arrows.

In Fig. 5 there is one object *anOrder* in front of its class *Order* and two *Item* objects *item1* and *item2* in front of the *Item* class symbol. Similarly to two-dimensional diagrams, each object has a *lifeline* and communication between the objects is presented by arrows from one lifeline to another, and the messages are layed out from top to bottom between the lifelines. Figure 5 shows the three-dimensional counterpart of the interaction diagram of Fig. 4. This diagram has been created from the class diagram in the background by clicking on the class symbols and by connecting the lifelines with message arrows.

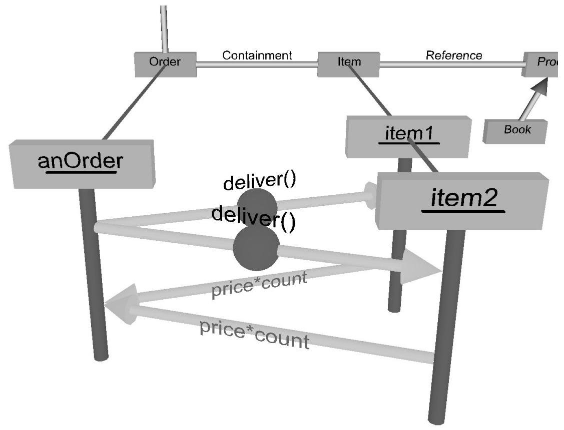

Figure 6: Sequence Diagram – Animated along Lifelines

4. ANIMATION OF INTERACTIONS

In the last section, we saw how interactions are created and manipulated in three dimensions. Here, we especially show how these interactions are animated. Our approach supports three kinds of interaction visualization:

Static Visualization: The most obvious kind is to show the objects, lifelines and messages exactly as they had been created (see Fig. 5). As with two-dimensional UML, this choice is adequate for *studying* interactions. But in addition to two-dimensional UML the user can walk around the model and examine it from different points of view.

Animation along the Lifelines: Again, the objects, lifelines and message arrows are placed where they had been created. But now the user can start the interaction by pressing a button. Then, a stimulus symbol moves along the first message arrow, starting at the sender. Within a predefined amount of time, the sphere arrives at the destination object lifeline and starts the next stimulus. This kind of visualization is adequate for *communicating* complex interactions to other modelers, who are familiar with UML interaction diagrams. Especially interactions where some messages are processed concurrently are addressed by this kind of visualization.

In Fig. 6, the two stimuli *deliver()* are processed concurrently. The two spheres have moved half of their way from the sender to the two receivers. During an animation of this kind, the labels of currently not processed messages are shown in gray.

Animated Objects: This kind of visualization helps the modeler to communicate her understanding of the real world to the customer. We expect only few customers to be familiar with interaction diagrams. Thus, we skip all the lifelines and message arrows and show only the objects. Again, a pressed button makes the first stimulus move. But this time it starts from the *center* of the sender object itself and moves to the receiver object's center, where it starts the subsequent stimuli.

5. CONCLUSION

We have presented our approach for an integrated way of developing systems: We use *interactivity* during analysis and design for rapid development of the static *and* the dynamic part of a system. We also want the user to examine a presented model interactively by walking around and starting different animated interactions. In contrast to this more individual way of examining a system, we sometimes want to direct the user's attention to a particular subsystem. Thus, we adopted the natural foreground-background metaphor to arrange objects in *three dimensions*. This is done when we lay emphasis on a subset of classes to visualize a certain static aspect as well as when we put animated interaction diagrams in front of the class diagram to stress the notion of dynamic behavior, which takes place within a static model. We extensively use *animation*: We have achieved a more natural visualization of interaction by animating the successive exchange of messages, and we use smoothly animated transformation when the visualization switches from one state to another thereby helping the user in tracing changes.

6. REFERENCES

[1] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, 1998.

[2] M. Deering and H. Sowizral. *Java3D Specification, Version 1.0.* Sun Microsystems, 2550 Garcia Avenue, Mountain View, CA 94043, USA, August 1997.

[3] Joseph Gil and Stuart Kent. Three dimensional software modeling. In *Proc. of the 20th Int. Conf. on Software Engineering*, pages 105–114. IEEE Computer Society Press, April 1998.

[4] Martin Gogolla, Oliver Radfelder, and Mark Richters. Towards Three-Dimensional Representation and Animation of UML Diagrams. In Robert France and Bernhard Rumpe, editors, *Proc. 2nd Int. Conf. Unified Modeling Language (UML'99)*, pages 489–502. Springer, Berlin, LNCS 1723, 1999.

[5] OMG, editor. *OMG Unified Modeling Language Specification, Version 1.3, June 1999.* Object Management Group, Inc., Framingham, Mass., Internet: `http://www.omg.org`, 1999.

[6] Tarja Systä. Automated support for constructing OMT scenarios and state diagrams in SCED. Technical Report A-1997-8, Department of Computer Science, University of Tampere, 1997. A-1997-8 is available as paper copy only.

[7] VRML 97 the virtual reality modeling language, 1997. `http://www.vrml.org/technicalinfo/specifications/vrml97/index.htm`.

[8] C. Ware and G. Franck. Viewing a graph in a virtual reality display is three times as good as a 2D diagram. In *IEEE Conf. on Visual Languages*, pages 182–183, October 1994.