Towards Three-Dimensional Representation and Animation of UML Diagrams

Martin Gogolla, Oliver Radfelder, Mark Richters
University of Bremen, FB3, Computer Science Department

Abstract

The UML notation is intended to be drawn on two-dimensional surfaces. However, three-dimensional diagram layout and animation may improve comprehension of complex diagrams significantly. The paper concentrates on special UML diagram forms well-suited for advanced visualization. It makes a proposal for representing and animating such UML diagrams in a three-dimensional style.

Keywords: UML Diagram, Visualization, Animation, Graph, Planar Representation, Three-Dimensional Representation.

A journey of a thousand miles must begin with a single step.

Lao-Tzu, The Way of Lao-Tzu.

1 Introduction

The Unified Modeling Language UML [BJR99] aims to become the industrial standard for software development and documentation. But also research projects consider the UML as a serious effort. Work is done in a large number of aspects ranging from questions of semantics to implementation issues.

Describing UML in an over-simplified manner, one could say that the language provides a number of different diagram forms for emphasizing different aspects of software. The diverse diagrams will be typically used differently in the various phases of software development. However, the documents [BJR99, p. 3-7] introducing the UML clearly point out the two-dimensional nature of UML diagrams:

> *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.*

Thus, all diagrams in UML and all examples given in the defining documents possess a two-dimensional nature. This paper supports the vision indicated above stating that three-dimensional layout should be studied. We see two main reasons for this:

(1) three-dimensional layout may *enhance* the comprehension of complex UML diagrams, and (2) three-dimensional layout may be *necessary* in situations where the two-dimensional representation is unable to express the intent of the UML diagram. For example, complex diagrams may be structured according to appropriate criteria by pushing uninteresting parts into the background, or graphs, which can only be displayed in the plane with intersecting edges, may be shown in three dimensions without intersection.

There is a number related works with a strong connection to the ideas presented here. One of our starting points was our experience with UML, its formal foundation, and some smaller example diagrams [GR98b, GR98a, RG98]. Animation of OMT scenarios has been studied in [Sys97]. The approach has similarities with the animation of interaction diagrams we propose. In [GK98, KG98] three-dimensional representation of diagrams is studied with respect to software engineering aspects. Information systems seem to be the main influence for the approach in [Koi92, Koi93]. Three-dimensional representation of diagrams and graphs is also well-known in the visual language and programming community [Shu88]. A basic vocabulary of three-dimensional representation elements was studied in [FGJ95], and the general benefits of lay-outing graphs in three dimensions was put forward in [WF94]. Three-dimensional diagrams were employed for distributed objects [VDS93, ZZ95], and algorithm animation [BN93],

The rest of the paper is structured as follows. Sections 2 to 5 study some particular UML diagram examples which benefit from a three-dimensional representation. The sections are devoted to class diagrams, object diagrams, animation of sequence diagrams, and three-dimensional sequence diagrams, respectively. The paper ends in Sect. 6 with concluding remarks and future work to be done.

2 Class Diagrams

We start with a very simple approach: We take a two-dimensional diagram, put it into three-dimensional space as a sheet (i.e., first we take only one layer), and then we push the uninteresting items into the background. Thereby, emphasis is laid on the important things in the foreground. The view may be changed by turning uninteresting things into interesting things and vice versa. The advantage of this approach is, that in three dimensions

- there is a natural notion of making things important, namely by drawing them closer to the viewer, and
- there is a natural notion of taking different perspectives, namely by moving around the scene in question and thereby watching it from different views.

Figure 1 gives an example of a non-trivial class diagram. Classes are shown as boxes, associations as thin connections, and generalization relationships as directed thick connections with the parent class positioned above the child class. Only a selection of the attributes and operations is shown. The class diagram is still moderate in size and describes aspects of a system for organizing and managing seminars. The original design can be found in [Bal96]. The classes on the right mainly deal with the seminars, the ones on the left with bookings for seminars. For seminars, (1) the class SeminarKind describes a general template, for example, a SeminarKind object may have the title Object-oriented Programming, and (2) the class Event characterizes concrete implementations, for example, the Object-oriented Programming events starting on 02-02-1999 and on 03-03-1999. Events are specialized to events open to the public and to closed events. For bookings, a distinction between specialized bookings for customers and bookings for companies is made. This reflects the different associated classes, namely Customer and PublicEvent for CustomerBooking, and Company and ClosedEvent for CompanyBooking.

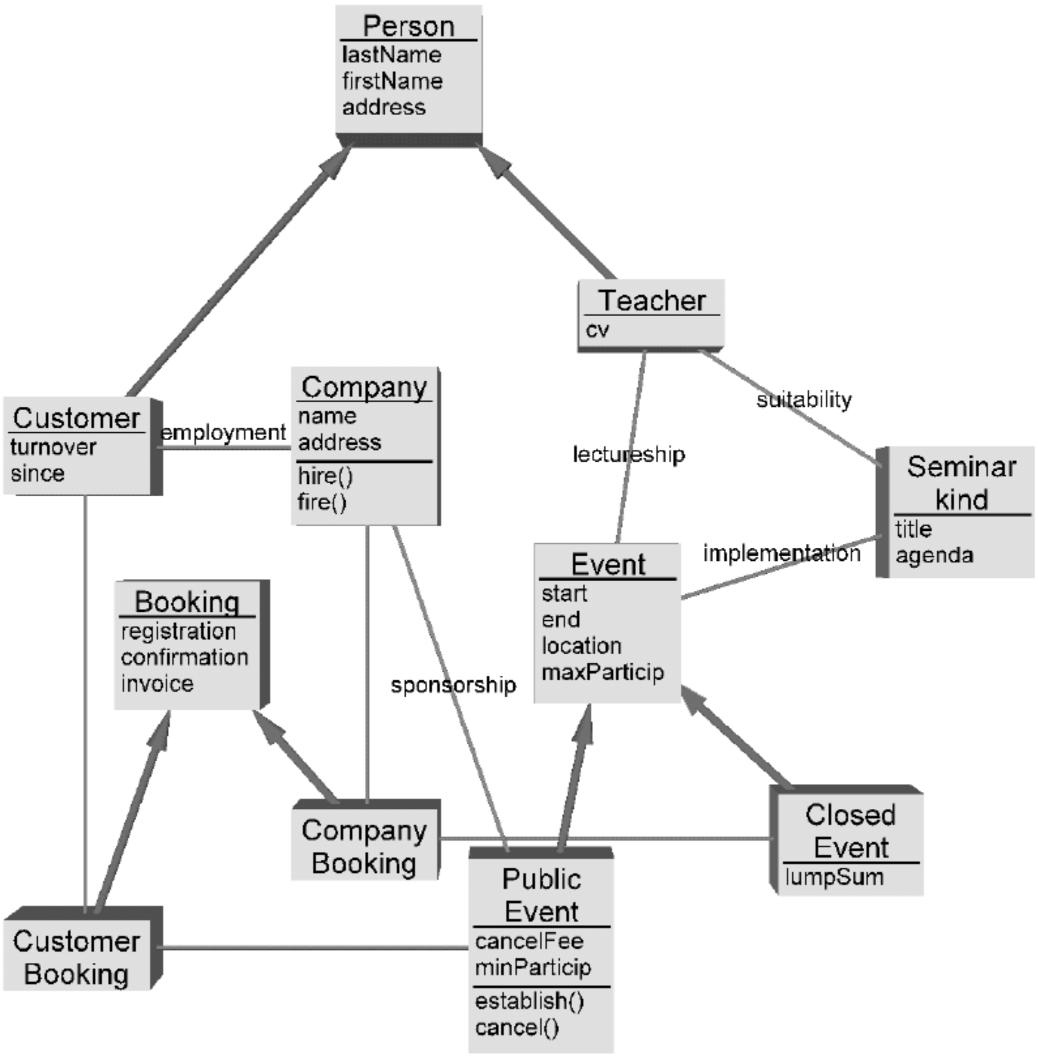

Figure 1: Complete Class Diagram for Seminar Organization

The classes in Fig. 1 are placed as boxes into space in a single plane, but they are viewed from a simple front perspective. In contrast, perspective and emphasis in Fig. 2 is different. Here, one concentrates on the booking aspect of the system by pushing the classes which are *not* important for that aspect into the background. Pushing classes into the background goes hand in hand with hiding the classes' details. Therefore the attributes and operations of unimportant classes are left out. In addition, one can now interactively take other perspectives than a front perspective or interactively rearrange the selected classes. In comparison to Fig. 1, the viewer in Fig. 2 has moved to the right. Changing views is also supported by smooth transitions between the different views, i.e., there is an animation from the first to the second view in which the respective classes are constantly moved until their desired position is reached. Thus the viewer of the animation does not perceive both views as different snapshots (as here in the static two-dimensional presentation of this paper), but by watching the animation, the interesting and important classes like the Booking class can be fixed and its position development can be traced. Analogously to highlighting the booking aspect, another view could emphasize the seminar aspect by focusing on and enlarging the right-side classes Teacher, SeminarKind, Event and its subclasses. Of course in general, any set of diagram elements could be selected to be focussed on. The expected benefit is increased clarity by embedding a small section of a class diagram into the entire system. This small example only demonstrates the principle idea of representing class diagrams in a three-dimensional way. The benefit will become more evident on much larger diagrams.

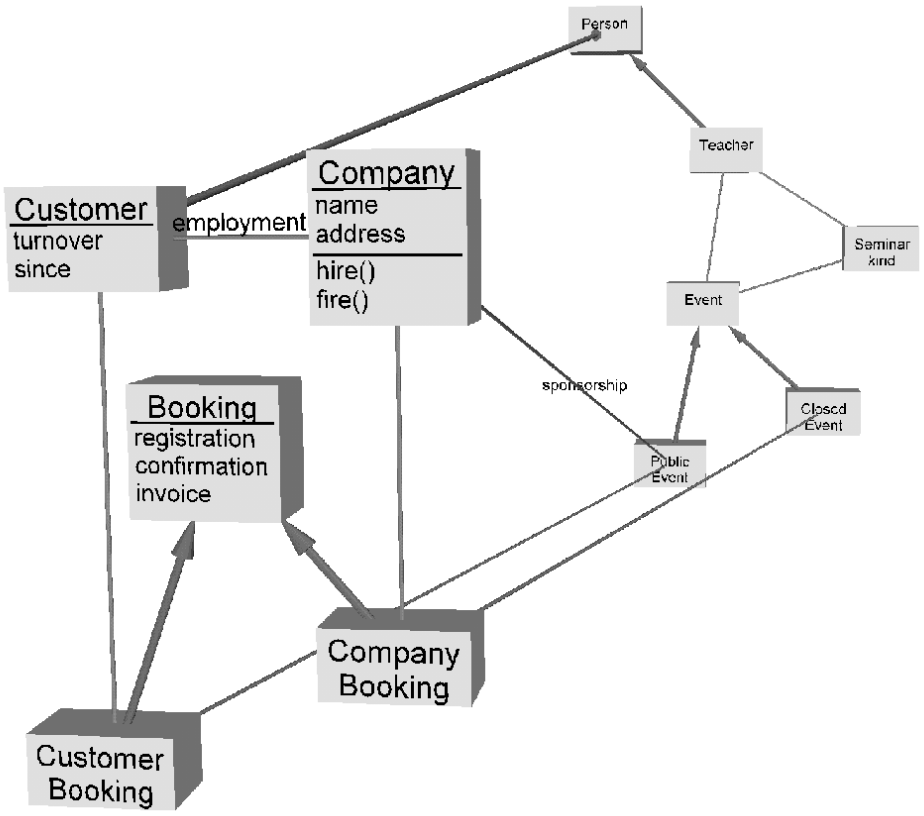

Figure 2: Class Diagram for Seminar Organization with Emphasis on Bookings

3 Object Diagrams

The next approach we want show is to take a two dimensional diagram and put the items of the diagram into three-dimensional space by some uniform criterion, for example by putting aggregates above the components and by choosing different planes for different kinds of objects.

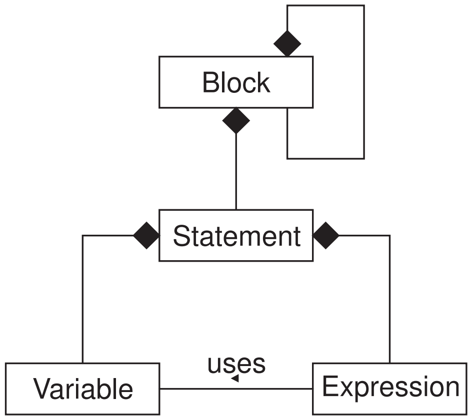

Figure 3: Class Diagram for Block-Statement-Expression

```
begin                      // Block B1
integer X, V1, V2; ...     // Statement S1, Variable V1, Variable V2
for ...                    //
  begin                    // Block B2
  integer V2, V3; ...      // Statement S2, Variable V2, Variable V3
  X:=V1*V2; ...            // Statement S3, Expression E3
  end; ...                 //
X:=V1*V2; ...              // Statement S4, Expression E4
if ...                     //
  begin                    // Block B3
  integer V3, V4; ...      // Statement S5, Variable V3, Variable V4
  X:=V2*V3; ...            // Statement S6, Expression E6
  end; ...                 //
end                        //
```

Figure 4: Sample Program to be Represented by an Object Diagram

Figure 3 shows part of a class diagram of a development environment for a programming language. The diagram mainly expresses that blocks may include statements and other blocks as components and that statements may involve variables and expressions as components. The association uses expresses that an expression may use a variable, e.g., by referring to its value. In Fig. 4, an unfinished program with various partly nested blocks, statements, variables, and expressions is given. Statements are variable declaration statements or assignment statements referring to previously defined variables.

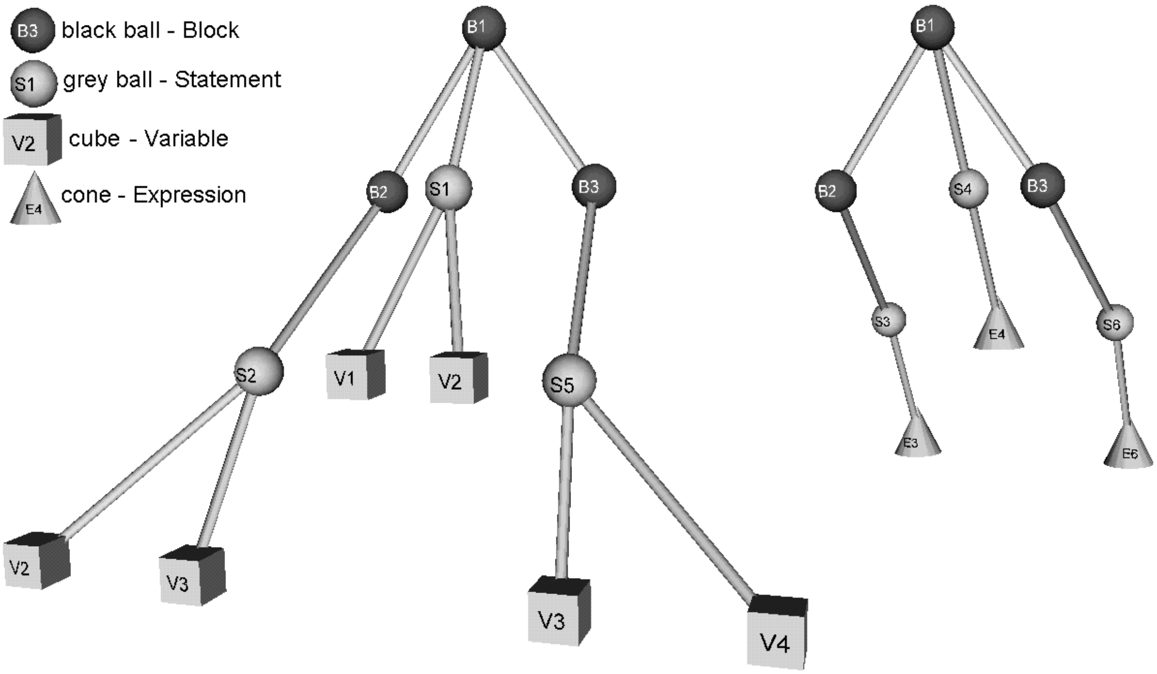

Figure 5: Left Part: Variable Declarations - Right Part: Expressions

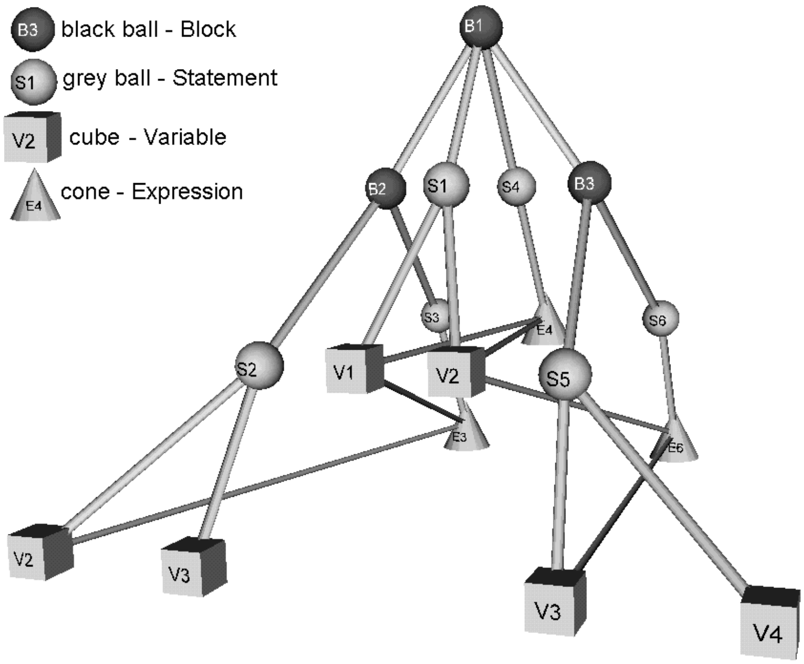

Figure 6: Links for Association uses

We now want to show that the example program of Fig. 4 may be represented as a three-dimensional object diagram fitting to the class diagram in Fig. 3. The object diagram will become a kind of pyramid. The pyramid was chosen because in this example we have two tree structures: one for the variables declarations and one for the expressions; one tree structure will become the front side of the pyramid, the other the back side. In Fig. 5 we have represented two initial parts of that object diagram: The left part displays the variable declaration statements whereas the right part covers the expression statements. The variable declaration statements lie in the front of the pyramid whereas the expression statements are positioned in the back of the pyramid. The uniform rule for displaying objects is that blocks are shown as black balls, statements as grey balls, variables as cubes, and expressions as cones. Blocks B1, B2, and B3 constitute the intersection of both parts. In both diagrams the aggregates lie above the components, and the aggregates are linked to their components by thick lines. Figure 6 finally puts the two parts mentioned before together and in addition shows with thin lines the uses links from the expressions in the back to the variables in front. For example, expression E6 is linked to (1) variable V2 declared in statement S1 and (2) variable V3 declared in statement S5. The edges representing the association uses are directed from the back to the front and are either parallel to the ground (variable declared in the same block) or climb upwards (variable declared in an enclosing block).

The three-dimensional object diagram strictly obeys the following rules:

- Display objects of different classes in different shapes or different colors (for example, variables as cubes and expressions as cones).
- Position aggregates above components (for example, blocks above statements).
- Arrange different kinds of objects in different planes (for example, variable declarations on the pyramid's front plane, expressions on the pyramid's back plane). Planes must not necessarily be disjoint.
- Show links between objects of different classes as edges connecting one plane to the other plane (for example, links for the association uses between the pyramid's back and front plane).

In this paper, we have to give an impression of the three-dimensional diagram by two-dimensional means. The three-dimensional nature is captured much better if one can analyze the diagram on a screen interactively by mouse movements, turn around the diagram and look at it from different perspectives. Thus, a user can generate her or his own animation as needed. This is possible, for example, with a simple prototype we have built which uses the Virtual Reality Modeling Language (VRML) [CG97, MC97] for visualization and animation.

4 Animating Sequence Diagrams

Up to now we have mainly taken one static UML diagram and have viewed it from different perspectives. Now our intention is different, because we want to point out dynamic aspects of one diagram by animating it. At the same time we would like to be able to move around the diagram. This section also explains the two orthogonal aspects of the title of the paper, namely the aspect concerning three dimensions and the animation aspect.

Figure 7 is taken from the UML notation guide [BJR99]. It shows the establishment of a telephone connection by means of a sequence diagram. Central to the diagram is the communication aspect between the three participating objects caller, exchange, and receiver. These three objects communicate by sending messages indicated as arrows directed from the sender to the receiver of the message.

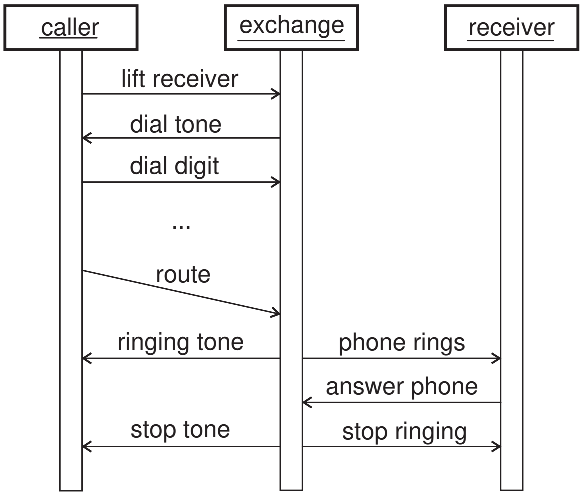

Figure 7: Sequence Diagram for Establishing a Telephone Connection

In Figs. 8 and 9, two snapshots from a three-dimensional animation of the diagram are captured. The three objects are represented as balls and are put into space on a single line. The connections between the balls show the communication channels. Messages sent from one object to another are displayed in the animation by a ball labeled with the message name. The ball moves from the sender object to the receiver object. The direction of the moving message ball is indicated in Figs. 8 and 9 by a grey arrow (the arrow does not appear in the animation and is used here only to indicate the move direction). The two snapshots are taken at different moments in the sequence, namely the lift-receiver and the ringing-tone/phone-rings moments. Also, two different perspectives are taken: The first perspective emphasizes the caller and the second one the receiver. Emphasizing means that the respective objects are put into the front of the scene. Thereby one process can be perceived from quite different viewpoints.

5 Three-Dimensional Sequence Diagrams

As a last example we want to explain how to employ three-dimensional diagrams for phenomena which are not expressible with two dimensions. It is quite well-known, for

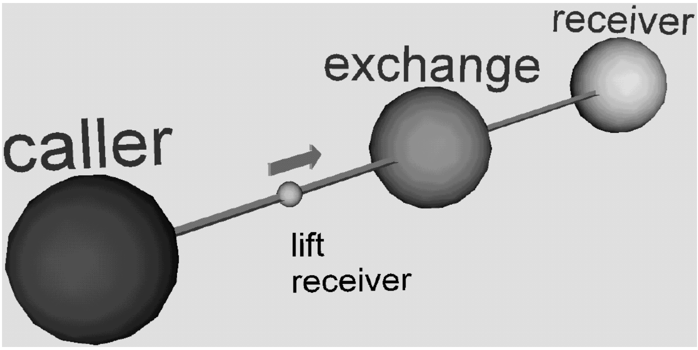

Figure 8: Snapshot 1 from an Animation for the Telephone Connection

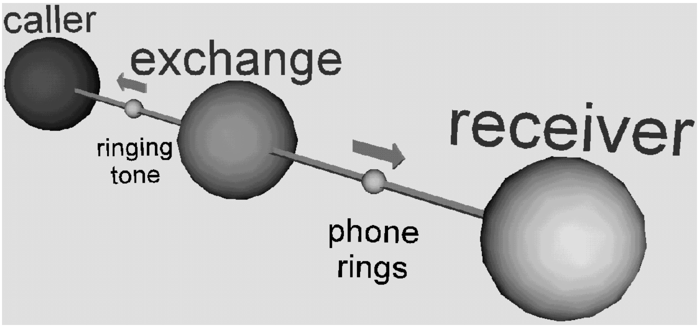

Figure 9: Snapshot 2 from an Animation for the Telephone Connection

example, that there are graphs which in the plane do not possess a representation without intersecting edges.

Sequence diagrams in UML are structured with respect to the participating objects. Each object in the sequence diagram has a lifeline, whereas communication between objects is presented by arrows from one lifeline to another lifeline. This means that it is easy and convenient to show communication between lifeline neighbors, but the representation of communication between non-neighbors becomes more involved. Figure 10 shows a sequence diagram with four simultaneous messages sent from one object to four other objects. In such a case, there is no easy to understand two-dimensional representation. Of course, this problem already occurs when one has to show three simultaneous messages. The work-around introduced in the diagram to express the simultaneous sending is a small curve. For example, the curve directly underneath the North object (together with the rest of the arrows) indicates that there are two messages, which go to the left and start at the same moment in the Control object: One message for the West object, the other for the North object. The curve is a feature not mentioned in the UML notation guide.

A way out of the dilemma is pictured in Fig. 11 where a three-dimensional representation of the diagram is shown. The lifelines are placed into space using different x- and y-coordinates. Simultaneous occurrence of communication can now easily be expressed, because in three-dimensional space it is possible that one lifeline has more

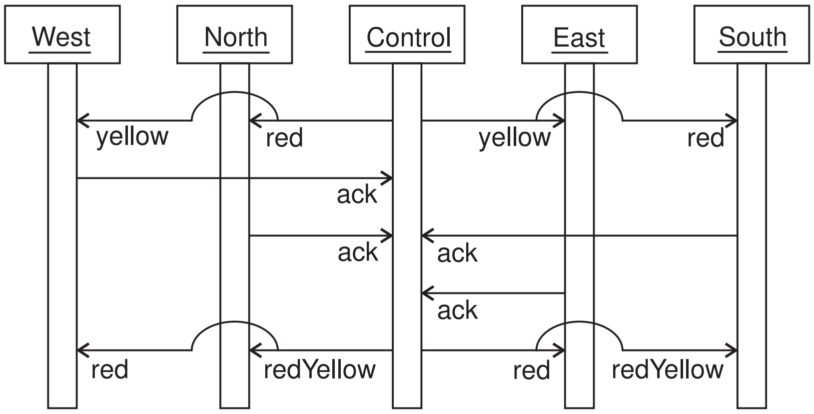

Figure 10: Sequence Diagram with four Simultaneous Messages

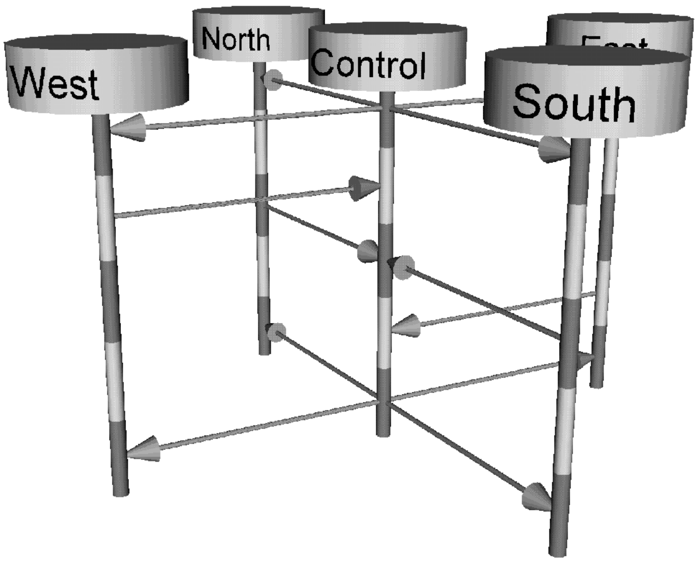

Figure 11: Three-Dimensional Sequence Diagram with Four Simultaneous Messages

than two neighbors.

6 Conclusion and Future Work

We have given some examples for UML diagrams which were designed to demonstrate that a three-dimensional diagram layout can enhance the comprehension of complex scenes. Our examples used class, object, and sequence diagrams.

As indicated in the very first quotation of the paper, we feel that the work presented here is only a first step on a journey. A lot of things remain to be done, both on the conceptual level and on the implementation level. Because UML is a rich language with respect to diagram forms, other kinds of diagrams and combinations of diagrams have to be studied:

- Use case diagrams with complex actor and use case relationships might benefit from three-dimensional features, for example, the position of actors in the diagram could reflect their position in the real-world system.
- Object diagrams could be looked at from different views, not only from the aggregation view we mainly used here, but also from a generalization or general association view.
- Activity diagrams with swimlanes can be very similar in their spatial requirements to the ones we mentioned for sequence diagrams.
- Three dimensions can also point out the relationships between different diagrams nicely. This is possibly because the different diagrams could take their own place in three-dimensional space, and relationships between diagrams can be indicated as connections in between like this has been done to some extent in [GK98, KG98].
- Object diagrams and their corresponding class diagrams, i.e., type and instance aspects, can be displayed by providing space for the classes and objects, and by indicating the type-instance relationship with connections. Layout of object diagrams in three dimensional space may follow various paradigms, for example, a real-life-space or real-life-time paradigm. Instead of zooming classes into the foreground (as described in Sect. 2), instances of classes could grow out of classes and come to the foreground where they form an object diagram which can be animated by a sequence of interactions.
- Sequence and statechart diagrams can be animated together in such a way, that the concrete message sequence turns out to be a run through the general and abstract scheme of the statechart.
- Our animation of sequence diagrams can also be understood as an animation of an equivalent collaboration diagram.
- Two complex sequence diagrams, which are closely connected, for example, through common objects, but which are too complicated to be shown in a single diagram, can be animated together. The simultaneous animation has the advantage, that only the active messages are shown. Thereby, the complexity can be reduced.

References

[Bal96] H. Balzert. *Lehrbuch der Software-Technik: Software-Entwicklung.* Spektrum Akademischer Verlag, Heidelberg, 1996.

[BJR99] G. Booch, I. Jacobson, and J. Rumbaugh, editors. *UML Summary (Version 1.3).* Rational Corporation, Santa Clara, 1999. http://www.rational.com.

[BN93] M.H. Brown and M.A. Najork. Algorithm Animation using 3D Interactive Graphics. In *Proc. 6th ACM Symposium User Interface Software and Technology (UIST'93)*, pages 93–100, 1993.

[CG97] R. Carey and B. Gavin. *The Annotated VRML 2.0 Reference Manual.* Addison Wesley, 1997.

[FGJ95] E. Freeman, D. Gelernter, and S. Jagannathan. In Search of a Simple Visual Vocabulary. In *Proc. 11th IEEE Symposium Visual Languages*, 1995.

[GK98] J. Gil and S. Kent. Three Dimensional Software Modeling. In *Proc. 20th Int. Conf. on Software Engineering (ICSE'98)*, pages 105–114. IEEE Computer Society Press, 1998.

[GR98a] M. Gogolla and M. Richters. Equivalence Rules for UML Class Diagrams. In Pierre-Alain Muller and Jean Bézivin, editors, *Proc. UML'98 Int. Workshop (UML'98)*, pages 87–96. ESSAIM, Mulhouse, France, 1998.

[GR98b] M. Gogolla and M. Richters. On Constraints and Queries in UML. In Martin Schader and Axel Korthaus, editors, *The Unified Modeling Language – Technical Aspects and Applications*, pages 109–121. Physica-Verlag, Heidelberg, 1998.

[KG98] S. Kent and Y. Gil. Visualising Action Contracts in OO Modelling. In *IEE Proceedings Software, 145*, pages 1–18, 1998.

[Koi92] H. Koike. Three-Dimensional Software Visualization: A Framework and its Applications. In *Visual Computing: Proc. Computer Graphics Int. Conf. (CG'92).* Springer, Berlin, 1992.

[Koi93] H. Koike. The Role of another Spatial Dimension in Software Visualization. *ACM Transactions Information Systems*, 11(3):266–286, 1993.

[MC97] C. Marrin and B. Campbell. *VRML 2.* Sams Net, Indiana, 1997.

[RG98] M. Richters and M. Gogolla. On formalizing the UML object constraint language OCL. In Tok Wang Ling, Sudha Ram, and Mong Li Lee, editors, *Proc. 17th Int. Conf. Conceptual Modeling (ER'98).* Springer, Berlin, LNCS 1507, 1998.

[Shu88] N.C. Shu. *Visual Programming.* Van Nostrand Reinhold Company, 1988.

[Sys97] T. 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.

[VDS93] J.-V. Vion-Dury and M. Santana. Virtual Images: Interactive Visualization of Distributed Object Systems. In *Proc. 8th OOPSLA*, 1993.

[WF94] C. Ware and G. Franck. Viewing a Graph in a Virtual Reality Display is Three Times as Good as a Two-Dimensional Diagram. In *Proc. 10th IEEE Symposium Visual Languages*, 1994.

[ZZ95] D.-Q. Zhang and K. Zhang. A Visual Programming Environment for Distributed Systems. In *Proc. 11th IEEE Symposium Visual Languages*, 1995.