Tools for Presentation and Animation of Dynamic Models in Object-oriented Design

Kurt Nørmark, Lars Iversen, and Per Madsen
Aalborg University
Denmark*

October 1, 1998

Abstract

Modelling the dynamic aspects of an object-oriented design is important to gain concrete insight into the ways objects interact and relate to each other. A similar insight may be obtained from a static model, but at a more abstract level and in a more indirect way. Dynamic models are usually presented as graph-based diagrams in which the vertices are objects and the edges are messages or relations among objects. In this paper we study such diagrams as the basis for tools in a dynamic modelling environment. We are especially interested in different ways to present time in such tools. One particular approach is to present "time by time", hereby leading to animation of dynamic models. By introducing the idea of 'design by animation' we aim at a radical improvement of the presentation and manipulation of dynamic models in contemporary CASE tools.

1 Introduction

Diagrammatic, graph-based presentations play an important role as the documentation of an object-oriented model. This is because of the documentation traditions in early phases of a software engineering process, and due to fact that graph-based models provide attractive and natural overviews of a set of objects together with their mutual relations [19].

Object-oriented models come in two different flavors: Static models and dynamic models. Static models are concerned with classes, methods, attributes, and relations among these. Dynamic models deal with objects, object interactions, and object relations. Some dynamic models provide for semantically complete specifications, whereas others are oriented towards examples.

This paper is based on the work with DYNAMO [17] in which we investigate the following overall hypothesis:

> *Programmers think in terms of objects, object relations, and object interactions during the creative phases of the design process.*

*Department of Computer Science, Fredrik Bajers Vej 7E, 9220 Aalborg Ø, Denmark. E-mail: normark@cs.auc.dk. WWW: http://www.cs.auc.dk/~normark/ — This research was supported by the Danish Natural Science Research Council, grant no. 9400911.

As a natural consequence of this hypothesis, the DYNAMO environment allows the designer to capture essential properties of an object-oriented model in concrete and tangible terms: objects and messages. In order to amplify the concreteness, DYNAMO is based on examples of object interaction in terms of scenarios.

The work with DYNAMO is oriented towards three research areas:

1. Definition of an object-oriented, dynamic modelling language.
2. Automatic generation of static models from dynamic models.
3. Construction of a set of tools for model building and exploration.

The modelling language is defined at an abstract level, and consequently a dynamic model is represented as an abstract syntax tree derived from an abstract grammar. The current version of the modelling language is described in details in a separate paper [16].

It is one of the basic ideas behind the work with DYNAMO that the dynamic model should be made early in the design phase, before (or perhaps simultaneous with) the creation of the static model. This is consistent with the ideas of 'use case driven design' [9]. We attempt to generate substantial parts of the static model from the dynamic models. As argued in [14], the majority of the information in a static model can be extracted from a set of scenario based dynamic models. A detailed account on this part of the DYNAMO work can be found in a separate paper [18]. The approach taken in this part of the work is to generate program outlines in a familiar programming language syntax. The program outlines are intended as *static overviews of a set of scenarios* rather than the first version of the implementation of the design. Alternatively, we could present the extracted static model as a class structure diagram.

This paper contributes to the third research area. The tools of the DYNAMO environment depend critically on diagrammatic means of presentation. In section 2 we introduce and discuss two well-known and widespread possibilities: Interaction diagrams [9] (also known as sequence diagrams [20], message sequence charts [27], or event trace diagrams [21]) and object-graphs (also known as collaboration diagrams [20]). In chapter 3 and 4 we elaborate on each of these kind of diagrams with the purpose to remedy a number of weaknesses and shortcomings, and in order to prepare for a more advanced specification of scenarios using animations.

2 Diagrammatic presentations of scenarios

A *scenario* is a concrete example of interaction among a number of objects. A scenario is described in terms of a message from the "surround" to a receiver object. The receiver, may in turn, send a number of messages to other objects, and so on recursively. Thus, a scenario is as a tree of messages which we often identify with the root node of the tree (the top-level message). In this paper we will call this structure a *message tree*.

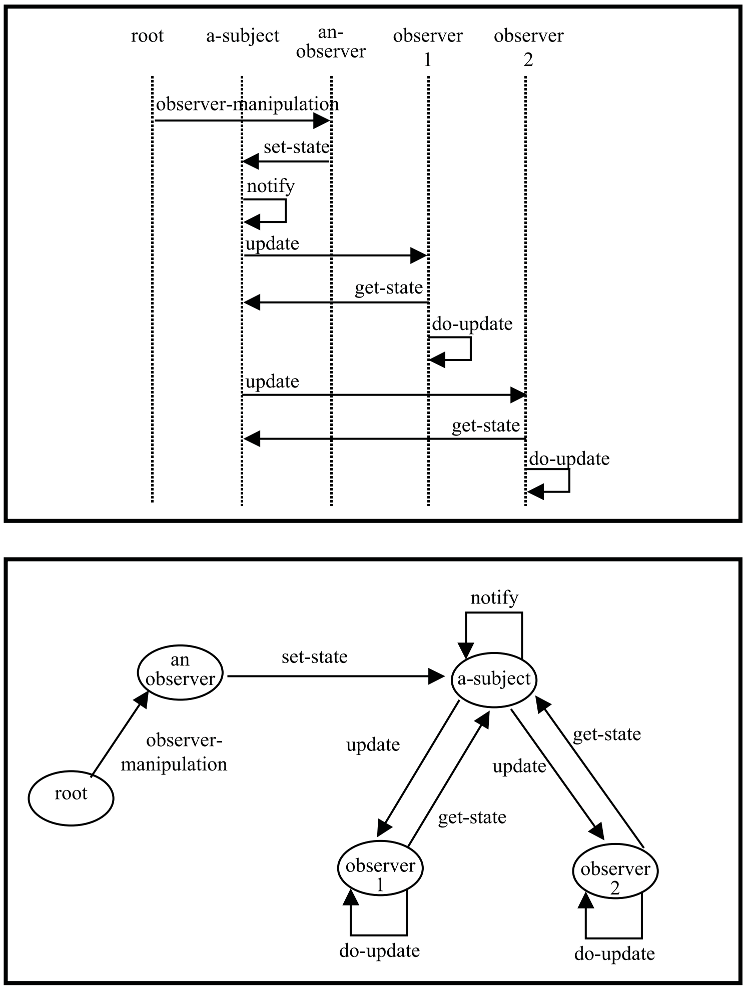

Figure 1: *An interaction diagram (above) and the similar object graph (below).*

There are two different scenario diagramming techniques in widespread use: Object graphs and interaction diagrams. In an object graph vertices represent objects and edges represent messages (or relations between objects via which it is possible to send messages). In an interaction diagram objects are represented by vertical lines, and messages are represented as arrows between the objects. Figure 1 shows comparable and stylized examples of the two kinds of diagrams. We will now discuss these two kinds of diagrams as the basis for tools in a dynamic modelling environment.

An object graph is probably the most intuitive notation. People who discuss an object-oriented design at a blackboard tend to use the object graph notation. As pointed out in [28], the designer is free to place an object anywhere in the two-dimensional space. This allows for natural grouping of related objects. As a contrast to interaction diagrams, an object graph is symmetric in its use of the two dimensions of the drawing surface. However, this can be regarded as

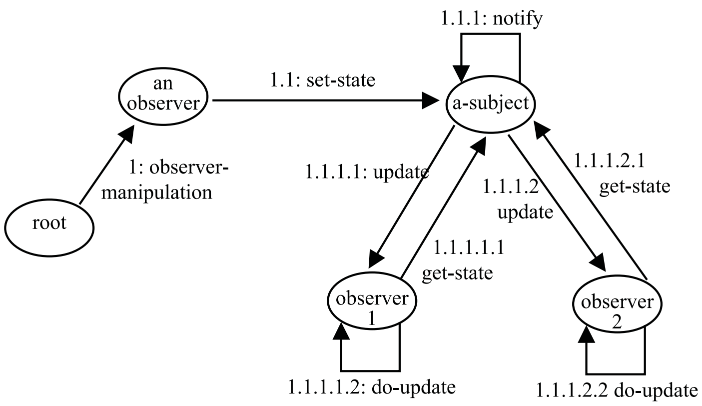

Figure 2: *An object graph with sequence numbers.*

a disadvantage from the tool perspective because it is inherently difficult to automatically make "natural layout" of graphs in a two dimensional surface [5].

It is possible to interpret an interaction diagram as an object graph. Following this interpretation the nodes of the graph are rendered as vertical lines, and the edges are horizontal arrows, or bended arrows from one node to itself. The node layout is linear and therefore trivial. Only the mutual, horizontal ordering of objects is an issue, perhaps with the goal of minimizing the distance between objects that communicate a lot.

The important and special characteristic of an interaction diagram is the significance attached to the relative vertical positions of the edges: The vertical dimension represents time. If timing aspects have to be shown in a more conventional object graph, additional graphical details in terms of sequence numbers need to be added to the graph. Figure 2 shows a version of the object graph from figure 1 with sequence numbers.

From a tool perspective it is relatively straightforward to draw an interaction diagram, primarily because there is no complicated layout problems that have to solved. However, as mentioned in [28], the number of objects that can be shown on the screen at a given point of time is lower than the number of objects that can be arranged in the two dimensional surface of an object graph. The opposite observation holds for the number of messages: In case there is heavy interaction among the objects (especially in case of multiple messages between pairs of objects) the interaction diagram is superior to object graphs.

The problem of "scaling up" is a major concern when discussing tools for presentation of dynamic models. We see three possible way of improving the presentation of large dynamic models:

- Subscenario abstraction.
- Presentation filtering.
- Alternative presentations of time.

Abstraction of subscenarios is a possible approach in dealing with scenarios that contain many objects and messages. In SCED [15] a subscenario may be abstracted to a single named interaction which represent all the detailed interactions in the subscenario. By defining part of a scenario to be a subscenario, the space limitation of interaction diagrams are alleviated in the vertical "message direction" as well as the horizontal "object direction". The reduction of space in the horizontal direction is due to the observation that objects, which are used only internally in the subscenario, do not need to be presented in the surrounding scenario.

The idea of *presentation filtering* is to suppress selected graphical details in order to provide for presentation of larger models. The more details we choose to present in diagrams such as in the figures 1 and 2, the smaller becomes the models that we can deal with on a screen or a piece of paper. In CASE tools it is attractive to support a variety of presentation filters which allows the designer to switch between detailed and overall presentations of dynamic models. Presentation filtering may be realized through a graphical zoom facility. Extreme graphical "zoom outs" have been described as *information murals* [10]. In the mentioned reference information murals are used to visualize huge interaction diagrams captured from running object-oriented programs.

In static presentations of dynamic models time is presented by various graphical means (described above). In such presentations both past and future objects, messages, and relations are shown relative to some given point of time. In more *dynamic presentations* of dynamic models, model time is represented by real time, as experience by the user of a tool (hereafter called *tool time*). As a function of tool time, objects, relations, and messages appear and disappear on the screen. In animative presentations the model elements are moved relative to each other as a function of time. We see animations as a new and interesting way of presenting dynamic models in the process of designing object-oriented software. Animative presentations lead to presentations of dynamic models in which we do not show legacy or futuristic elements, and as a consequence we may be able to present larger models than in similar static presentations.

As it appears from this discussion there are relative strengths and weaknesses with both interaction diagrams and object graphs. In the following we will first discuss and elaborate on interaction diagrams as the basis of a scenario editing tool in the DYNAMO environment. Following that we will return to the object graph as the basis of a *design by animation tool.*

3 Presentations based on interaction diagrams

The primary internal representation of a scenario in DYNAMO is the message tree in which a node N represents a message to some receiver object O. In DYNAMO we assume that every object has a name. Sons of N represent submessages from O to other objects (or to O itself), and so on recursively.

An interaction diagram is structurally different from a message tree. The difference between the two is that an object only appears once in an interaction

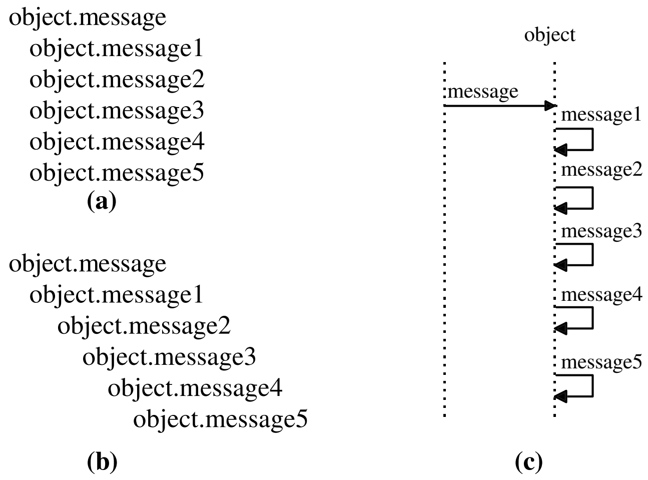

Figure 3: *Two different message trees* (**a**) *and* (**b**) *with identical interaction diagrams* (**c**).

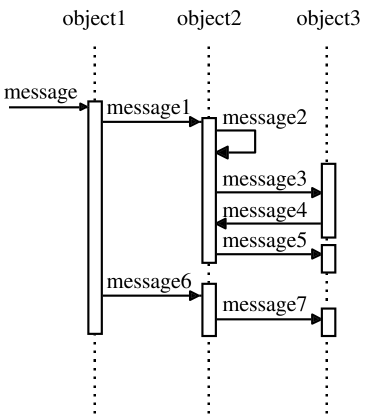

Figure 4: *An interaction diagram in "Booch style".*

diagram, but it may appear several times, at several levels in a message tree (as instances of a name representing the object). As an implication of this an interaction diagram may be *ambiguous* relative to the underlying message tree. Figure 3 shows an extreme example of two different message trees which cannot be distinguished in the usual rendering of an interaction diagram.

The ambiguity problem pointed out above is formulated as a problem of visualizing methods by Wolber in [29]. As a reaction to this problem (among others) Wolber proposes a new graphical formalism called a *use case structure chart*. A use case structure chart is a hierarchical method decomposition tree

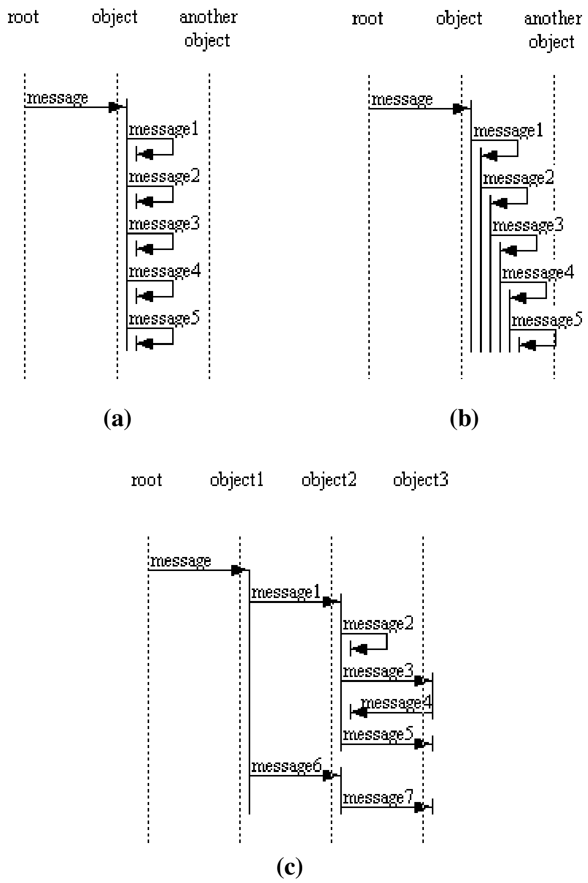

Figure 5: *DYNAMO interaction diagrams of message tree* **a** *and* **b** *from figure 3 and a diagram similar to the "Booch style diagram" of figure 4. The interaction diagrams are taken directly from the DYNAMO environment.*

(which also supports some special means for event handling).

As illustrated in figure 4 several authors, such as Booch, impose a vertical box on the object line representing the method, which is invoked by a message [2]. In case several methods are activated on a single object it would be natural to nest the "method boxes" into each other, but this is graphically intractable because of the narrow width of the method boxes.

In order to remedy the ambiguity problem we have designed a new graphical notation in the DYNAMO project in which each method activation is shown as an *activation line* parallel with the object line, but with a few pixels offset to the right (see figure 5). The principle is that each activation of a method M on an object O gives rise to a new activation line on O. In more graphical terms, each arrow head points at a new activation line. In uni-sequential models (models without multiple, active objects) a message does not return before all the sub-messages have returned. In such situations it is always the case that

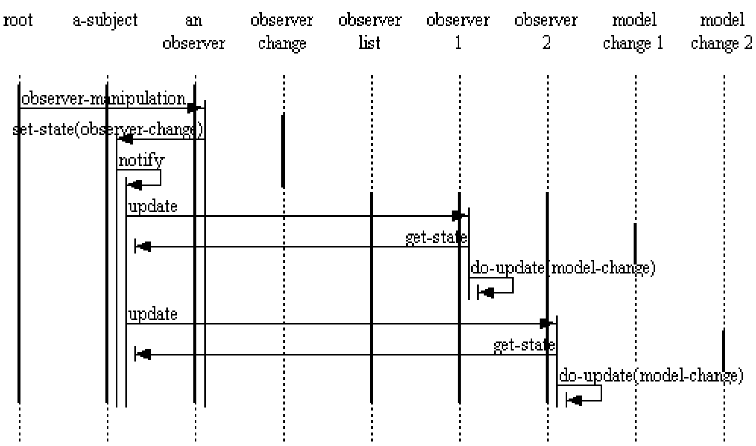

Figure 6: *An interaction diagram for the observer design pattern with object relevance lines (in black and white only).*

inner activation lines of O are temporally contained in outer activation lines of O.

Interaction diagrams with activation lines show the message tree of the underlying scenario in an unambiguous way. It is possible to track the stack of method activations by following the edges from one activation to another, possibly in a number of different objects. Figure 5 illustrates DYNAMO interaction diagrams for the two message trees in figure 3(a) and (b), and a diagram similar to the one in figure 4.

Objects can appear and disappear during a scenario. It is important that this dynamic aspect can be reflected in the presentations of the model. Object appearance may in some situations be equivalent to object creation, but in others it may signal that an object becomes relevant for the scenario we are specifying. In the DYNAMO modelling language it is possible to specify when an object becomes relevant via the so-called *object provision mechanism* [16].[1] Also the period of time in which the object is relevant for the design is dealt with in DYNAMO. Some objects are local to an activation (and thereby cease to exist at method return time), others exist as long as at least one relation associates them to other living objects.

It is important to reflect graphically the object provision time and the period of object relevance. In DYNAMO interaction diagrams each object is presented as a dashed object line, which is "open" in the positive time direction. We superimpose a closed, colored line segment on the object line which shows the period of time the object is relevant for the purposes of the scenario. We call

[1] An object provision states that an object, with some given properties, exists at 'object provision time'. The object is either created at object provision time, or it has been created at an earlier time. When an object is provided, the object enters the scene in a 'magical way'. For the purpose of a particular dynamic model we do not care about the detailed object creation circumstances.

this line segment an *object relevance-line*. Different colors are used to signal the role[2] of the object relative to the enclosing scenario. It should be noticed that the object relevance line is contained in the *object life line* (the time span between object creation and object destruction), as defined in UML [20]. Figure 6 shows an example of an interaction diagram with object relevance lines.

It is, of course, possible to add additional graphical details to an interaction diagram, hereby making it possible to illustrate more and more aspects of the dynamic model directly through the interaction diagram. However, the more details we add the more difficult it becomes to grasp and extract the general overview of the message structure. The UML diagrams for dynamic modelling [20] contain several examples of fairly detailed graphical (and embedded textual) notations which make it difficult to comprehend the diagrams. In the DYNAMO environment we use graphical popup boxes[3] and a variety of browsers with textual fields to present and edit the details of a dynamic model (described in [16]). In addition we use presentation filtering, as described in section 2, to add or to suppress certain details of the diagrams.

Given the intuitively appealing notation of an interaction diagram, it is important to extend this also to cover the way we edit an interaction diagram. In the work with DYNAMO we have made a direct manipulation interface for the following editing operations on interaction diagrams:

1. Creation of a new message.
2. Changing the receiver of a message.
3. Changing the place from which a message is sent.
4. Creating a new object in a particular interaction context.

When a designer wants to add a new message to a scenario he or she drags a new arrow from the sending object to the receiving object. New objects can be created via a popup menu. Objects are created relative to the interaction context in terms of the currently selected message and the currently selected object.[4] The name and the parameters are typed directly into the diagram (no indirection in terms of a data entering windows is present). Similarly, the designer can change the receiver by dragging the message arrow to another object; And by dragging the initial point of an arrow, the underlying message subtree can be moved from one sending context to another.

Changing the receiver of a messages supports *redistribution of responsibilities* among the objects in a dynamic model. Changing the place, from which a message is sent supports *revised implementations of existing responsibilities*. It

[2]The *object roles* include global/persistent objects, objects provided in a parameter list, object provided in the body of a method, and objects provided as part of a message result.

[3]A graphical *popup box* appears when the user rests a little while with the mouse button pressed down on some graphical item which represents a message or an object in the interaction diagram.

[4]Let us give a couple of examples of the use of the *interaction context* when creating new objects. If an existing object O is selected it is possible to create an object associated with, or part of O. If an existing message M is selected it is possible to create an object local to the method underlying the message M, or an object which is result of sending the message M.

is a typical refinement step in a design to decide that a responsibility should be changed from one object to another. Consequently it is important to visualize a *holistic view* of the responsibilities among the objects. Interaction diagrams and object graphs are good in that respect - much better the traditional static descriptions (class diagrams or source program) in which we show fragmented and information-limited pictures of a design. Furthermore it is important to provide flexible and natural means of interactions for changing the responsibilities. The direct manipulation interfaces described above seems well suited to support a flexible design process.

The objects and messages in a dynamic model are tightly related to each other, either directly or indirectly. Some relationships are shown clearly in an interaction diagram. As an example the relationship between the sender and the receiver of a message is shown by an arrow. Other relationships are more difficult to extract from an interaction diagram. As examples of these we can mention (1) the relationship between a message and its local objects, (2) the relationship between a message and the object possibly returned as the result of passing the message, (3) the relationship between a message and the objects playing the roles as parameters of the message, and (4) the associations and aggregations between objects. It is important to come up with ways to show such relationships. In DYNAMO we temporarily highlight related objects when a message or an object is selected. The highlighting disappears when the mouse button is released. Using temporary highlighting, we keep the diagram free from disturbing details. Furthermore, we utilize the dynamic nature of a computer screen as the medium being used to present a dynamic model.

As already pointed out above there is a fairly large body of literature on various aspects of scenarios and static presentations of scenarios, such as interaction diagrams [9, 20, 27, 21, 28]. We have found that the work closest to DYNAMO is that on SCED [15, 14, 13]. The scenario concept (called algorithmic scenarios) in SCED is relatively powerful, featuring both repetition and conditionals. As already mentioned earlier in this paper SCED supports interesting abstraction mechanisms on scenarios. The diagrammatic notation of scenarios in SCED is relatively rich because it includes specific graphical renderings of informal comments, states, actions, repetitions, conditionals, and subscenarios besides the fundamental interaction diagram notation of objects and messages/events. The most important result of the work with SCED is, however, an algorithm for automatic generation of state machines from a set of scenarios [12]. The algorithm is based on a relatively old algorithm by Biermann et al. which deals with program construction from examples [1].

Interaction diagrams can be used for other purposes than (object-oriented) design. Scene [11] is an example of a system which is able to produce interaction diagrams from program executions. The main focus in Scene is to use scenarios for understanding and browsing existing software. As such, scene represent a "reversed approach" compared with SCED and DYNAMO.

4 Animations based on object graphs.

Both interaction diagrams and object graphs provide static views of a dynamic model. In section 2 we have already discussed relative advantages and disadvantages of the two kinds of diagrams, and in section 3 we introduced a number of interactive features on top of static presentations of interaction diagrams. If, however, we want radical changes in the way we visualize and manipulate dynamic models we must improve on the handling and the representation of time in dynamic models.

Recall that interaction diagrams devote one of the two dimensions of the drawing surface to a representation of time. As such, time is represented in a clear and dominating way. If there are many objects in an interaction diagram it can be necessary to scroll the diagram window horizontally. Horizontal scrolling in an interaction diagram is very unfortunate, because it severely obstructs the general overview of the model. Another priced paid for the "time dimension" is the lack of natural grouping means among objects presented in an interaction diagram.

In object graphs time is represented by graphical adornments in terms of sequence numbers on the edges of the graphs. As illustrated in figure 2 this is a less dominating and quite compact representation of time. It is, however, difficult to track the temporal aspects of a dynamic model from the sequence numbers. Casual studies of an object graph with sequence numbers is not likely to reveal potential timing difficulties in a dynamic model.

The idea is now to represent model time by means of tool time. In this context, *model time* represents the relative ordering of messages (and possibly other actions) in a dynamic model. As mentioned earlier in the paper *tool time* refers to the designer's experience of time when using the modelling tool.

This representation of "time by time" leads to diagrams that change appearance as a function of time. As the amount of diagram changes is relatively small over a short period of time the diagrams are apprehended as *animated diagrams*. In principle both interaction diagrams and object graphs can be used as a starting point of an animative presentation. However, due to (a) the intuitive appeal of the object graphs as claimed in section 2, (b) space economical use of the two dimensional drawing surface, and (c) the weak representation of time in static object graphs, it seems obvious to use object graphs (without sequence numbers) for animation purposes. In case we used animated interaction diagrams, time would be presented both graphically and temporally. Clearly, this would be an overkill. Thus, in the following discussion we will focus exclusively on animated object graphs.

Animated object graphs are used in a process which we call *design by animation* [6]. In a design by animation process we *create* and *explore* dynamic models represented as scenarios. It is our basic hypothesis that a design by animation process gives ideal support for the object-oriented designer in the creative phases of the OOD process. Moreover, we are convinced that the result of a design by animation process provides attractive documentation of the dynamic aspects of an object-oriented design, perhaps even for people with little insight in object-oriented modelling concepts. Design by animation depends critically on the computer used as a *dynamic medium.* This should be seen as a contrast to the use of static mediums (like paper). As can be seen from the first sections of this paper it is very difficult to capture the inherent dynamic properties of a dynamic model on a static medium.

In a typical scenario only a small number of messages are passed at a given point in time. In uni-sequential models, only a single message is passed at a time. Similarly, in some dynamic models many objects have limited lifetimes. The node representing an object with limited lifetime appears at a given point in time and it possibly disappears before the completion of the animation. Also the relations (associations) among the objects are dynamic elements. Thus, when dealing with animated object graphs we are only faced with a fraction of the messages, objects, and relations at a given point in time. This makes it possible to handle larger models. Consequently, animation contributes in a positive way to alleviating "the problem of scaling up", which we discussed earlier in section 2 of this paper.

The animation of objects and messages may in the extremes be either discrete or continuous. *Discrete animation* may be understood as a sequential presentation of a number of (partial and developing) object graphs. Using discrete animation we show a sequence of partial object graphs that illustrate the temporal development of the underlying model. A discrete animation may be understood as a sequential presentation of object graph "snapshots". Each snapshot only presents the model elements that are relevant at a given point in time. Neither "history elements" nor "future elements" are part of the actual snapshot.

Using *continuous animation* the neighboring snapshots of a discrete animation are "interpolated" by means of smooth and sliding movements of the graphical elements in an object graph. It is one of our hypotheses that continuous movements make it easier and more natural to track the model changes between two points in time. This is confirmed by Statsko in his work on animation of algorithms [25]. In addition, we want to assign semantic significance to the ways objects and messages are animated in XMAS [6, 8, 7]. The "tool time" dependent animation carries more information than just the relative ordering of actions in the underlying dynamic model. The interval of "tool time" between two snapshots may vary depending on the amount of animation which is required to bring us from one "model time" to the next.

In order to make the discussion concrete, we will now describe the most important patterns of animation used in XMAS:

- **Object creation**: If an object A is responsible for creation of object B, the B object grows out of A. The new object B moves from the inner of A to its final destination on the screen, and while moving its size increases from half to full size.

- **Object provision**: In case an object B is provided, we do not know for sure which object created it in the past. This is animated by dropping B from the top of the window and after that moving it to its final destination in window.

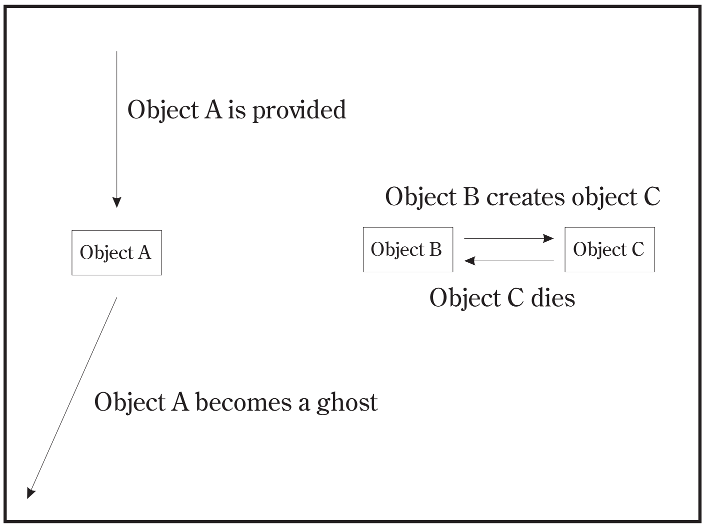

Figure 7: *The animations used to create, provide, and "ghostify" an object.*

- **Object deletion**: When an object no longer is accessible (because, for instance, it goes out of scope) it may be deleted from the screen. Object deletion is animated in the opposite way of object creation and provision. Consequently objects either disappears into the object that created it, or they move into a corner of the screen, where they become "ghost objects".[5]

- **Message passing**: A message m is passed from one object A to an object B by running and arrow out of A towards B. During the animation a textual label appears, showing the name of the message together with actual parameters. The return of the message is shown by retracting the message from the receiver, hereby moving the arrow back into the sending objects. During this animation a description of the returned value appears as the label of the message arrow.

Figure 7 and 8 illustrate these animations, although, of course, it is difficult to make convincing illustrations of animation on a piece of paper. [6]

Although well researched, the graph layout problem remains as one of the severe difficulties in dealing with object graphs. In other words, it is not easy to determine the most natural and the most optimal position of nodes in the two-dimensional drawing surface. The difficulties are amplified by aesthetic requirements (such as minimizing crossing edges) and hidden semantic requirements (such grouping of related objects). In the XMAS research we have until

[5]An object becomes a ghost if we have incomplete knowledge of the object's life time. This may occur in case a provided object goes out of scope. The reason is that a provided object is created before its provision, but we don't know exactly where in the model it happened.

[6]The XMAS tool can be downloaded from the World Wide Web [7] for a more interactive experience.

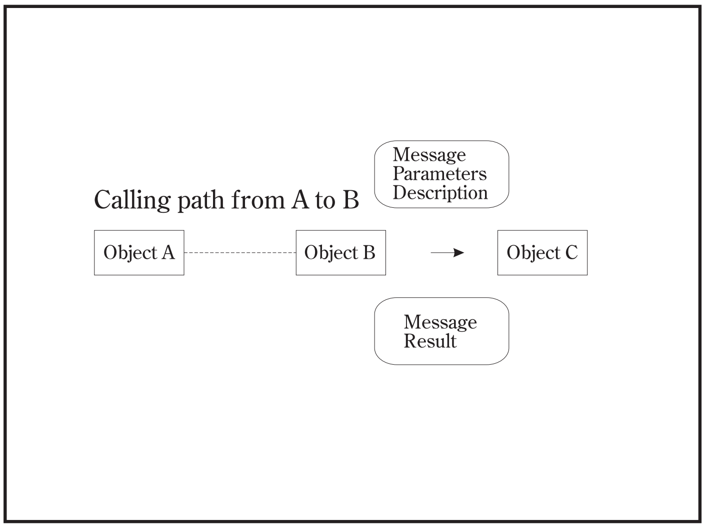

Figure 8: *The animations used for message passing.*

now ignored these difficulties by placing objects manually on the screen. In a more practical and complete tool, however, this is not an appropriate solution.

In order for XMAS to be a complete *design by animation tool* we also need to edit a dynamic model in terms of animation. To this endeavor, it is tempting to gain some inspiration from film editing. We map the message tree into a one-dimensional *reel* by a depth-first traversal of the tree. While a reel normally consists of equally spaced frames, an XMAS reel is made up of a hierarchy of *scenes*. In terminology of the message tree, a scene consists of a number of neighboring siblings in the tree, including the descendants of those siblings. Thus, a scene corresponds to a structurally well-defined portion of the message tree.

To edit the reel we use navigation primitives such as *GotoFirst, GotoPrevious, GotoNext* and *GotoLast*. In XMAS, these navigation primitives are provided in both normal speed and high speed versions, see figure 9. Using the high speed versions in both forward and backward direction it is possible to retain some kind of holistic overview of the scenario, despite the fact that only a subset of the objects and messages are presented graphically at a given point in time. We also support editing primitives like *Cut, Copy* and *Paste*. Besides these, a number of direct and animative manipulation primitives would be useful to create and modify messages and objects. However, in the current version of XMAS we use a simpler approach. Instead of direct and animative editing operations we create and modify the dynamic model by means of a number of dialogue boxes (one box for object creation, object provision, message passing, etc).

Modifying the reel via one of the above mentioned primitives implies the risk of an inconsistent reel. An inconsistency can for instance occur if we delete a sub-scenario that creates an object, which is referred at a later point in the

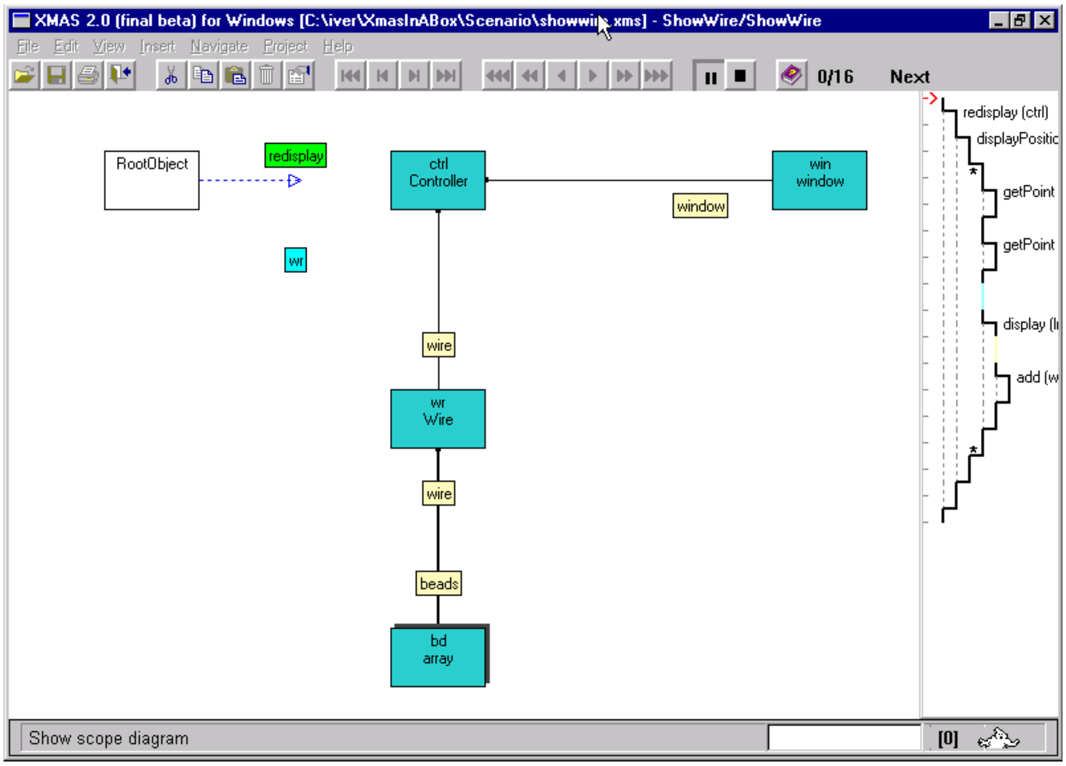

Figure 9: *A snapshot of the XMAS tool. The animation of the object graph takes place in the large area to the left. The diagram to the right, called a box diagram, provides a compact static overview of the dynamic model.*

model. In this situation it will be difficult to visualize a message to the non-existing object. Such (temporary) inconsistencies are well-known from structure editing in general, but they may be especially severe in our context. The reason is that the inconsistencies can make it difficult to animate the model, such as navigating to the place where the inconsistency can be alleviated. In XMAS, measures have been taken to ensure that all editing operations deliver a consistent model. In some situations this is achieve by adding elements, or ignoring changes in the internal "runtime" tool representation of the model. Such temporary counter measures are not reflected in the real and underlying dynamic model.

Editing and navigating a dynamic model via exclusive use of an animative presentation may be too difficult in practice. As already mentioned, the problem is to maintain an overall understanding of the entire model via the animated presentation. Therefore we find it useful to propose an supplementary, static presentation of the dynamic model besides the animative presentation. An interaction diagram is well suited for this purpose, because that kind of diagram in our opinion presents time in the most *graphically* comprehensible way, while at the same time providing a general overview of the dynamic model. In XMAS we use a simplified an compact diagram called a *block diagram*, which shows messages and levels of interaction, but no objects, see figure 9.

There is only little literature on "proactive application" of animation relative to the implementation of a design. The book *Software Visualization* [26] contains papers on a number of relevant and similar works. Algorithm animation is one such neighbor area which is covered extensively in the book [3, 4]. Some general animation techniques is also described [25].

Shilling and Statsko have made a system called GROOVE [23, 24], which is based on animation in a similar way as XMAS. GROOVE can be used in an object-oriented design process as well as for visualization of a running program. In GROOVE, the dynamic model is presented side by side with elements of the static model. GROOVE is primarily oriented towards design and run-time visualization of C++ programs. In comparison, no bindings to a particular programming language is found in DYNAMO or XMAS.

In addition to the work on GROOVE, Salmela et al. have described some ideas of animating real-time object-oriented software [22].

5 Status and conclusions

In this paper we have discussed two particular diagrams as the basis for presentation and editing of dynamic models. We have been concerned with the scale up problem, the problem of clear and unambiguous presentations of dynamic models, and the problem of presenting time in a natural way.

If we want to present time graphically, we recommend the use of interaction diagrams. We also recommend a variant of interaction diagrams, which shows the message tree in an unambiguous way. The details of such a variant has been described in this paper.

If we want a temporal presentation of time, we recommend the use of object graphs instead of interaction diagrams. This leads to animated diagrams, and it suppports a 'design by animation' process. The main challenges in dealing with animated diagrams are to come up with genuine patterns of animations, and to allow for natural ways of editing an animation. We have proposed specific animation patterns in this paper. With respect to editing of animations, our work is still in an early phase.

References

[1] A. W. Biermann and R. Krishnaswamy. Constructing programs from example computations. *IEEE Transactions on Software Engineering*, SE-2:141–153, 1976.

[2] Grady Booch. *Object-oriented analysis and design with applications, second edition.* The Benjamin/Cummings Publishing Company Inc., 1994.

[3] Marc H. Brown and John Hershberger. Fundamental techniques for algorithm animation displays. In John Stasko, John Domingue, Marc H.

Brown, and Blaine A. Price, editors, *Software Visualization - Programming as a multimedia experience*, chapter 7, pages 81–102. The MIT Press, 1998.

[4] Peter A. Gloor. User interface issues for algorithm animation. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization - Programming as a multimedia experience*, chapter 11, pages 145–152. The MIT Press, 1998.

[5] Ioannis "Yanni" G. Gollis. Graph drawing and information visualization. *ACM Computing Surveys*, 28A(4), December 1996.

[6] Lars Iversen and Per Madsen. Design by animation. Master's thesis, Department of Computer Science, Aalborg University, Denmark, June 1998. In Danish.

[7] Lars Iversen and Per Madsen. The WWW home page of the XMAS project. http://www.cs.auc.dk/∼normark/xmas/, June 1998.

[8] Lars Iversen and Per Madsen. XMAS - experimental modelling with animated scenarios (in danish). Master's thesis, Department of Computer Science, Aalborg University, Denmark, january 1998.

[9] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Overgaard. *Object-Oriented Software Engineering – A Use Case Driven Approach*. Addison-Wesley Publishing Company and ACM Press, 1992.

[10] Dean F. Jerding, John T. Stasko, and Thomas Ball. Visualizing message patterns in object-oriented program executions. Technical Report GIT-GVU-96-15, Georgia Institute of Technology, may 1996.

[11] K. Koskimies and H. Mössenböck. Scene: Using scenario diagrams and active text for illustrating object-oriented programs. In *Proceedings of the 18th international conference on software engineering, Berlin, Germany, March 1996.*, March 1996.

[12] Kai Koskimies and Erkki Mäkinen. Automatic synthesis of state machines from trace diagrams. *Software - Practice and Experience*, 24(7):643–658, July 1994.

[13] Kai Koskimies, Tatu Männistä, Tarja Systä, and Jyrki Tuomi. SCED - an environment for dynamic modeling in object-oriented software construction. In Boris Magnusson et al., editor, *Proceedings of the Nordic Workshop on Programming Environment Research, NWPER'94, Lund*, pages 217–230, 1994.

[14] Kai Koskimies, Tatu Männistä, Tarja Systä, and Jyrki Tuomi. On the role of scenarios in object-oriented software design. In Lars Bendix, Kurt Nørmark, and Kasper Østerbye, editors, *Proceedings of the Nordic Workshop on Programming Environment Research, NWPER'96*, pages 53–69. Department Computer Science, Institute for Electronic Systems, Aalborg University, R-96-2019, May 1996 1996. http://www.cs.auc.dk/∼normark/-NWPER96/proceedings/proceedings.html.

[15] Kai Koskimies, Tatu Männistä, Tarja Systä, and Jyrki Tuomi. SCED: A tool for dynamic modelling of object systems. Technical Report A-1996-4, Department of Computer Scienece, University of Tampere, Finland, 1996.

[16] Kurt Nørmark. Towards an abstract language for dynamic modelling in object-oriented design. In Raimund Ege, Madhu Singh, and Betrand Meyer, editors, *Tools 23*, pages 120 – 131. IEEE Computer Society Press, 1997.

[17] Kurt Nørmark. The WWW home page of the DYNAMO project. http://www.cs.auc.dk/∼normark/dynamo.html, 1997.

[18] Kurt Nørmark. Synthesis of program outlines from scenarios in DYNAMO. Submitted for publication, 1998. Available from http://www.cs.auc.dk/∼normark/dyn-models/static-models/synpooutl.pdf.

[19] Stephen North. Visualizing graph models of software. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization - Programming as a multimedia experience*, chapter 5, pages 63–72. The MIT Press, 1998.

[20] Rational Software Coorporation. UML notation guide 1.1. Available from http://www.rational.com, September 1997.

[21] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-oriented Modeling and Design.* Prentice-Hall International, 1991.

[22] Marko Salmela, Marko Heikkinen, Petri Pulli, and Reijo Savola. A visualisation schema for dynamic object-oriented models of real-time software. In Boris Magnusson et al., editor, *Proceedings of the Nordic Workshop on Programming Environment Research, NWPER'94, Lund*, pages 73–86, 1994.

[23] John J. Shilling and John T. Stasko. Using animation to design, document and trace object-oriented systems. Technical Report GIT-GVU-92-12, Graphics, Visualization, and Usability center, Georgia Institute of Technology, 1992.

[24] John J. Shilling and John T. Stasko. Using animations to design object-oriented systems. *Object oriented systems*, 1(1):5–19, September 1994.

[25] John Stasko. Smooth, continuous animation for portraying algorithms and processes. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization - Programming as a multimedia experience*, chapter 8, pages 103–118. The MIT Press, 1998.

[26] John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price. *Software Visualization - Programming as a multimedia experience.* The MIT press, 1998.

[27] International Telecommunication Union. Message sequence chart. Technical Report Z.120, International Telecommunication Union, 1996.

[28] Kim Walden and Jean-Marc Nerson. *Seamless Object-Oriented Software Architecture - Analysis and Design of Reliable Systems*. Prentice Hall, 1995.

[29] David Wolber. Reviving functional decomposition in object-oriented design. *Journal of object-oriented programming*, 10(6):31–38, October 1997.