Technical Report 98-09: Visualizing Adora Models

Stefan Berner, Stefan Joos, Martin Glinz
Dept. of Computer Science, University of Zurich
Winterthurerstr. 190, CH-8057 Zurich
Switzerland
{berner, sjoos, glinz}@ifi.unizh.ch

Martin Arnold[1]
EDS/FIDES Informatik
Postfach, CH-8036 Zurich
Switzerland
martin.arnold@fides.ch

Abstract

Most current object modeling methods and tools have weaknesses both in the concepts of hierarchical composition and in the visualization of these hierarchies. Some methods do not support hierarchical composition at all. Those methods which do, employ tools that provide explosive zoom as the only means for the visualization of hierarchies.

In this paper we present an approach for the visualization of hierarchical object models that is based on the notion of fisheye views. This concept can display local detail and global context of a view in the same diagram, thus allowing a user to easily navigate in hierarchical structures without offending the principle of abstraction. We introduce the ideas behind the concept, illustrate the zooming mechanism and sketch the algorithm for the implementation of this kind of zooming. The work presented here is part of an effort to create a new method called Adora[4] for object modeling that provides strong support for hierarchical composition/decomposition.

1 Introduction

1.1 Graphical Requirements Models

Graphical requirements models follow a long tradition in the field of requirements elicitation and specification. The basic idea is to specify requirements for a software system through a model that describes structure, functionality and behavior of a software system. Graphical models claim to be easy to understand, especially for non-specialists, and easy to maintain. There exists a broad variety of graphical models, e.g. data flow diagrams, state transition diagrams and statecharts [Hare87], petri nets, entity relationship models and nowadays object-oriented models like those of [Booc94, Cha+93, Jac+94, OML96, Rum+91, Sel+94, UML97, Wir+90] etc. The purposes of these models range from simple graphic visualization up to achieving specific effects, for example reduction of complexity.

1 Work was done when the author was with the University of Zurich

1.2 Model Visualization

Using computer systems and software tools, graphical models can be visualized by displaying and manipulating the model on a display device using a graphic notation. A *view* is a part of a model to be displayed. Visualization through views requires *navigation*. Navigation has two aspects, a *cognitive* and a *mechanical* one. The cognitive aspect deals with the mental effort to locate the current focus[2] or to move it. The mechanical aspect relates to the mechanical effort (e.g. movement of a mouse) to achieve a cognitive navigation goal.

The cognitive, non-mechanical effort for navigational activities is called *cognitive overhead* [Bea+90]. A good visualization concept is critical both for understandability and ease-of-use of graphical models. A good concept should:

- support *orientation* in the model by visualizing as much local detail as needed without losing the global context of the focused elements
- minimize the *cognitive overhead* for navigation
- increase *expressiveness* by including the semantics of the model in the visualization
- foster *understandability* by supporting the abstraction mechanisms of the model

In order to assess the current state of visualization concepts of graphic requirements modeling tools, we have analyzed three groups of tools (table 1).

Tool-Type	Tool-Name	Method/Language [reference]
Scientific prototype	KOGGE Tool BONSAI	KOGGE is a CASE-Tool-Generator; BONSAI supports the OO-Method BON [Koe+96]
	SOM - Tool	Semantic Object Modeling [Fer+94]
	Macrotec	Business process modeling with macronets/petri nets [Kel+95]
	EGS1	Functional modeling, executable graphical specification, data flow diagrams [Gas+95]
Commercial Net-Modeling and Simulation-Tools	SystemSpecs	Petri net simulation/animation [SysS96]
	Statemate	Statecharts modeling and simulation/animation [Stat96]
	MicroSaint	Discrete event modeling and simulation [Micr96]
	Arena	Discrete event simulation with SIMAN/CINEMA-Tools [Aren96]
Commercial software development environments [Dart87] and/or CASE-Tools [Fugg93]	ObjecTime® Toolset	ROOM-Method [ObjT96]
	Rational Rose	OOA/D-Method [Rose96]
	Software through Pictures StP	Multi method; OMT [Rum+91], OOA/D [Booc94], ... [StP96]
	Objectory	OOSE [Objy96]
	Innovator CASE Workbench	OMT, OOSE, ... [Inno96]

TABLE 1. System modeling tools and supported methods/languages

2 A focus is a location in a visualized model which is of momentary importance for the user [Sar+92].

Most tools operating on flat or practically flat models support only scaling in order to handle large sized models. Few tools have map windows for orientation and roam or scroll bars for navigation [Bea+90, Kaen96].

Tools operating on hierarchical model structures normally visualize a single node with its direct successors in one view. A few tools visualize all nodes in one view (for example Statemate). Some tools of this kind offer scaling possibilities, map windows or hierarchy-overviews (for example SystemSpecs) to manage the complexity of big models. Most tools operating on hierarchical data structures offer just *explosive zooming*, which means that a zoomed node will explode entirely in the existing or in a new window with all its direct successors. As a consequence, these tools offer either *global context without local detail* views or *local detail without global context* information in the same view.

Global context and local detail in one view are realized in very few tools, full flexibility in scaling and zooming is *not* offered at all. Compared with the essential modeling tasks the cognitive overhead increases too much when models become bigger.

1.3 Motivation and Basic Ideas

In this paper, we present a visualization concept for an object modeling language with a strong mechanism for hierarchical model decomposition and part-of-abstraction. We consider hierarchical decomposition of models with the semantics of a part-of-abstraction to be crucial for the understandability and maintainability of large models (that means for the normal case in industrial practice). As none of the existing object-oriented requirements modeling languages fully support such a mechanism[3], we are developing a language and method (Adora[4]) which does[5]. Our work on visualization is based on this language.

The principal idea of our approach is to apply the notion of fisheye views [Furn86] to the visualization of requirements models. We define fisheye views with multiple foci for navigating in hierarchically clustered networks of objects. View generation is model driven. It follows the structure and abstractions of the model. The concept integrates local detail and global context in a single view, which eases orientation and navigation in the model, thus minimizing the cognitive overhead.

As our concept allows less interesting elements to be visualized on an abstract level together with the details of some elements of special interest, we have strong capabilities for supporting abstraction mechanisms in the object model that is being visualized and thus foster the expressiveness and understandability of the model.

---

3 Most existing object-oriented requirements modeling languages either do not fully support a hierarchical model decomposition which employs a *part-of-abstraction* at all (e.g. [Shl+88]) or they provide only a clustering mechanism with weak semantics (e.g. [Rum+91] [Jac+94] [Booc94] [UML97]). Few approaches do provide such a decomposition mechanism ([Cha+93] [Wir+90]). However, they encounter modeling anomalies which are due to using class models instead of object models [Joo+97].

4 ADORA is an acronym and stands for *Analysis and Description Of Requirements and Architecture*.

5 An introduction to this language is not the focus of this paper.

1.4 Organization of this Paper

Before we discuss the basics of visualization and our visualization concept in detail, it is necessary to introduce the kind of models to be visualized. Since we visualize Adora models, section 2 gives a brief overview of the Adora language.

In section 3 we discuss and differentiate the concepts of scaling and zooming for navigation and introduce the notion of fisheye views.

In section 4 we describe our visualization concept. Subsection 4.2 begins with a discussion of the requirements that led to our design. In the subsections 4.3 we present our concept of 'model-driven' fisheye views. We concentrate on zooming which is the core mechanism for handling these views and give a concrete example in subsection 4.4.

Section 5 sketches our zooming algorithm.

The paper concludes with a short discussion of achievements and of the state of work.

2 A Brief Overview of the Adora Language

Our work on visualization of hierarchical models was motivated by the need for an adequate visualization mechanism for Adora[4] but it is adaptable for similar languages, too. Adora is a semiformal, object-oriented method for requirements modeling which is currently being developed at the University of Zurich. In this section, we give a brief overview of Adora concepts. In particular, we sketch the hierarchical structure of Adora models.

The basic idea of Adora is to model the aspects of data, functionality and behavior in a *single* hierarchical object framework. Modeling is based on objects instead of classes. Thus, we resolve modeling anomalies that occur in decompositions of class models [Joo+97].

Hierarchical decomposition and part-of-abstraction is a key feature of Adora. Compositions are not simply clusters with no or weak semantics. A composition in Adora is a first class object having full object semantics including structural relationships and behavior. These structural relationships as well as the behavior of a composition are true abstractions of the relationships and behavior of its components. The semantics of behavior decomposition is based on an extension of the statechart [Hare87] mechanism to objects [Glin93, Glin95].

Figure 1 gives an example of an Adora model. It shows a view of a requirements model for controlling a double elevator. This example illustrates some important features of Adora models with respect to visualization. (1) The view contains compositions on various levels of detail. For example, Control Panel is shown with all its components. Left Cage is displayed without components (three periods following the name denote it to be a composition). The components of Right Cage are displayed in full detail with exception of Inside Door. (2) Left Cage and Right Cage are different objects of the same type. When this situation occurs in a view, the type of these objects is annotated in square brackets. (3) Relationships are visualized on the same level of abstraction as their corresponding objects. For example, between

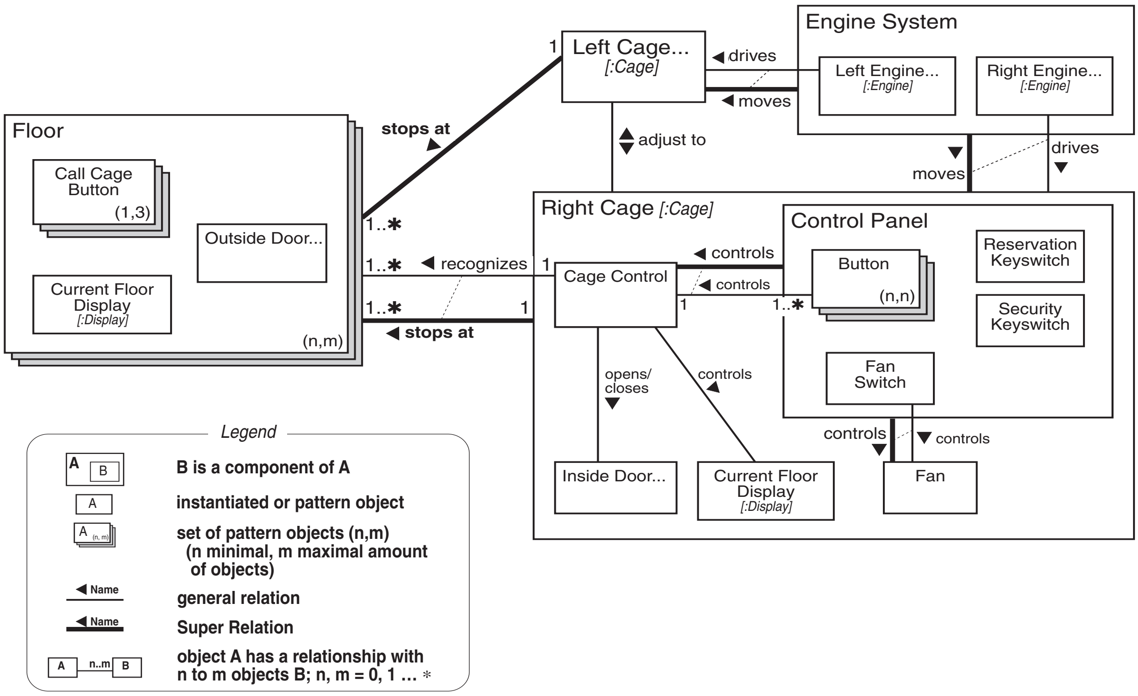

FIGURE 1. System structure of a double elevator

Left Cage and Floor there is only an abstract relationship stops at, because Left Cage is displayed without its components. For Right Cage, there is also the component relationship recognize from Cage Control to Floor visible because Right Cage is displayed in full detail.

We use the same style of visualization for hierarchical behavior models. For the sake of brevity, we have omitted behavior modeling from our example.

The most important feature of Adora in the context of this paper is the notation for nested hierarchical structure including the ability to represent objects and their relations on different levels of abstraction in a single diagram.

The underlying data structure to be visualized in our approach is a kind of 2D network of components and relationships. Each component is part of exactly one composition and compositions can be higher level components as well. Therefore our object model can resemble the Higraph structure [Hare88] or hierarchically clustered networks [Sch+96]. When generating views, this explicit hierarchy of objects will be used by the visualization algorithm (see section 5) to determine how detailed each element will be displayed.

The definition of the Adora language is yet not fully complete. However, the principles of the language are stable enough that we can derive the requirements for an adequate visualization of Adora object hierarchies.

3 Basic Terms

3.1 Navigation in Hierarchical Models

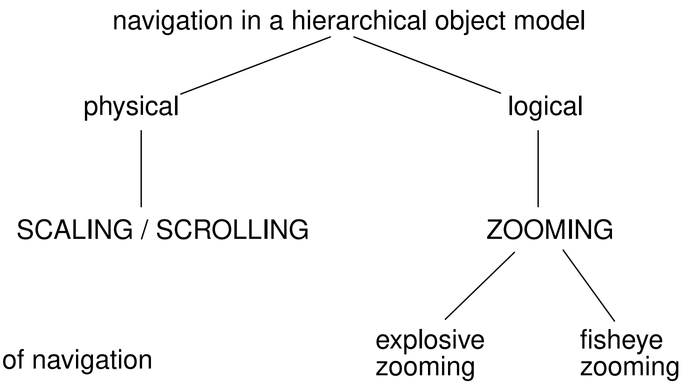

FIGURE 2. Types of navigation

When we visualize hierarchically structured models, we differentiate between two distinct types of navigation, a physical and a logical one (see figure 2)

Physical navigation is necessary when the actual visualized model (view) exceeds the size of the available display. This kind of navigation requires the ability to *scale* the size of the visualized model by shrinking or growing the size of the visualized elements. With *scrolling* or with the use of a map window [Bea+90, Kaen96] we can change the displayed part of a view. Physical navigation is well known as it is the normal way of navigation in flat models.

As the model has a hierarchical structure and the visualization should allows us to show different hierarchical levels of the whole or of a part of the model, we must be able to navigate through the hierarchy. *Logical navigation* in a hierarchical structure means that we need to find the actual position of a local element in the global context of the hierarchy, or that we would like to change the foci of certain visualized elements. To handle this type of navigation adequately, we *zoom-in* or *-out*. Zooming-in means that more details of a deeper hierarchical level will be visualized. Zooming-out means that a more abstract view of the selected elements will be produced. Using *explosive zooming*, the global context gets lost, while the zoomed node will explode entirely in the existing or in a new window with all its direct successors/child-nodes.

Fisheye zooming produces a local detail view while preserving the global context in the same view. A fisheye view [Furn86] is an information visualization concept, which combines and balances local detail and global context together into a single view. As the term indicates, the fisheye view concept is based on the idea of a fish's eye which works like a wide-angle lens. The idea is to show local detail (the objects of interest to the user) in full, while displaying successively less detail for information being further away from the focus.

There exist several fisheye concepts for visualizing large information spaces. The differences between them are mainly the type of data structure they are able to display and the dependencies on their application semantics. An extension to the traditional fisheye view concepts allows to show multiple areas of interest in detail simultaneously [Sch+96]. This means, for example that zooming into more than one composition in an object model will result in a view with *multiple foci*.

4 Visualization Concept

4.1 Context

The tool environment for Adora (which we are currently developing) consists of a repository, a graphical model editor and a model animator/simulator. The general architecture is conventional and straightforward: All functional tool components are grouped around the repository. We will not discuss the general architecture here and concentrate on the visualization concept for the graphical model editor and animator/simulator. This leads us to the core requirements concerning the visualization, which form the basis of our concept.

4.2 Requirements

We analyzed the intended use of the tool and set up navigational scenarios. From these scenarios (S1 – S3), we derived core requirements for our visualization concept. The most important ones (R1 – R7) can be found below.

Navigational Scenarios

- S1 The user wants to select one (S1.1), many (S1.2) or all compositions (S1.3) to be displayed more detailed, which means to explode their structure and visualize their components and relations.
- S2 The user wants to select one (S2.1), many (S2.2) or all exploded compositions (S2.3) to be displayed less detailed, which means to hide/abstract from their internal structure and visualize their components and relations.
- S3 The user wants to change the size of the visualized elements without changing logic structure of the view (and amount of information) which is currently displayed. This means to shrink or grow all elements.

Core Requirements

- R1 The degree of detail must be freely selectable (S1, S2).
- R2 A view should provide local detail and global context (S1, S2, see also section 1.2).
- R3 Displayed objects must not overlap (S1, S2, see also section 1.2).
- R4 Temporal scrolling activities must not distort the view (S3).
- R5 Zooming and scaling are each independent user tasks, which can be combined in a free order (S1, S2 vs. S3).
- R6 It must be possible to set up an arbitrary number of foci simultaneously in a view (S1.2, S1.3, S2.2, S2.3; please note, excessive use of multiple foci of course increases the cognitive overhead. However, in several scenarios it is convenient to see the details of co-operating objects but not details of their context. This results in the necessity to

keep at least two or three compositions exploded. This constraint lead us to (R6)).

R7 It must be possible to select one, many or all compositions in a view to be focused on or to be removed from the focused elements (S1, S2).

4.3 Concepts

The basic principle for our ‘model-driven’ adaption of fisheye views is to generate views according to the decomposition structure of the model and its abstractions.

Navigation

Logical navigation based on explosive zooming is not suitable for our purpose because the global context of an object gets lost when a zoom step is performed. A view in our concept represents the *whole* model. The degree of detail and thus the level of abstraction may vary, but a view *always* shows the whole context.

In order to minimize the cognitive overhead for navigation (see section 1.2) and to fulfill (R6), multiple foci must be allowed in the same view. So, the visualized degree of detail of each object can be freely adjusted, independently from other objects (R1). To navigate through the model, fisheye zooming and scaling/scrolling is possible. Zooming is the technique for changing the focus and the degree of detail of a view. Scaling operates on a view as well but the logical structure of the view remains unchanged. Scrolling operates just on the displayed part of a view and also does not change the logical structure of a view.

Zooming

Zooming changes the visualized part of the model structure in a view. Zooming takes advantage of the hierarchical structure and explicit decomposition of the model to allow views with different levels of abstraction. These abstractions are ‘real’ abstractions because they are not based just on the omission of language elements but on the utilization of an explicit model decomposition and part-of-abstraction, respectively.

- By *Zooming-In* the user selects a specific element as a (new, additional) focus meaning that he wants to see the components of this composition. Zooming-in is a stepwise uncovering of the underlying structures.
- By *Zooming-Out* the user removes a specific element from the set of defined foci meaning that he wants to see less detail than before. Zooming-out of a selected composition leads to an abstraction of the underlying structure of this composition. The selected composition will be shown as a black box, hiding internal structures, behavior and sub-relations etc.

Zooming can be done by selecting an element in the model, by fully qualifying the name of the element to be zoomed or by selecting an element in a map window displaying the hierarchy structure. In any visualized configuration, each composition can be focused on; this means that each composition can be zoomed-in or zoomed-out independently of the visualized context surrounding this composition. With this idea we realize *multiple foci.*

Scaling and Scrolling

Scaling of the elements in a view preserves the currently visualized structure with respect to the existing foci while changing the size of all elements by the same scaling factor.

- Scrolling preserves proportions. Only the displayed part of a view will be changed.
- Scaling up/down elements in a view may result in displaying/omitting some language elements, for example names of object attributes, event names etc. The omission of language elements of a specific type, e.g. object attributes, relation names etc. clarifies a view but allows only limited abstraction, merely on a 'syntactic' level, because scaling cannot exploit an appropriate model decomposition to gain abstraction and abstract views, respectively.

Scaling and scrolling are well known and widely implemented. Therefore we confine ourselves to a more detailed description of the zooming mechanism, which we currently are implementing in a tool prototype.

4.4 Example

The starting point for the illustration of the zooming concept is the situation presented in figure 3: The double elevator system is visualized with its top level compositions Floor, Left Cage, Engine System and Right Cage as black boxes. The objects currently displayed in the view are marked by filled black circles in the following figures. The example will illustrate a sequence of zoom-in and zoom-out requests to demonstrate view generation and navigation. The magnifying glass indicates the composition which will be zoomed.

In the example, we proceed as follows: We zoom-in until all objects of the hierarchy (represented in the tree as nodes) are visible (same view as in figure 1). Finally one zoom-out step is performed. The first step is a zoom-in request on the Floor composition. The Floor composition will be exploded and its internal structure will become visible.

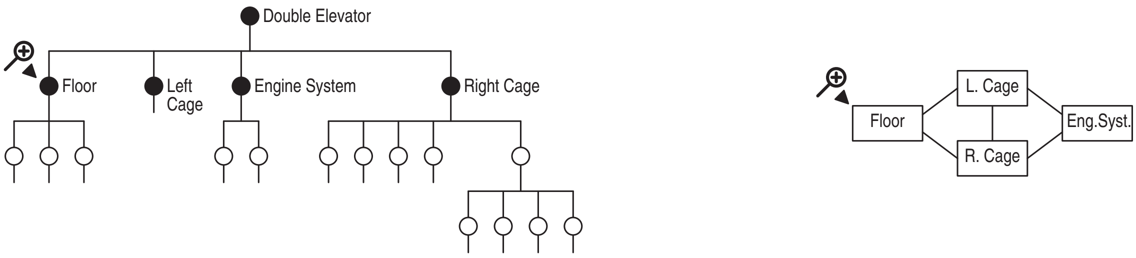

FIGURE 3. Zoom-in request on the Floor composition; the left side the figure shows the hierarchic structure of the double elevator, and the right side the corresponding view. A black node on the left side denotes a visible object whereas a white node denotes a hidden object.

Figure 4 shows the view after a zoom-in on the Floor composition has been performed. Floor is shown with its direct successor components Call Cage Button, Current Floor Display and Outside Door. The next step is to set two additional foci; first to the Right Cage composition and then to the Control Panel composition inside the Right Cage composition.

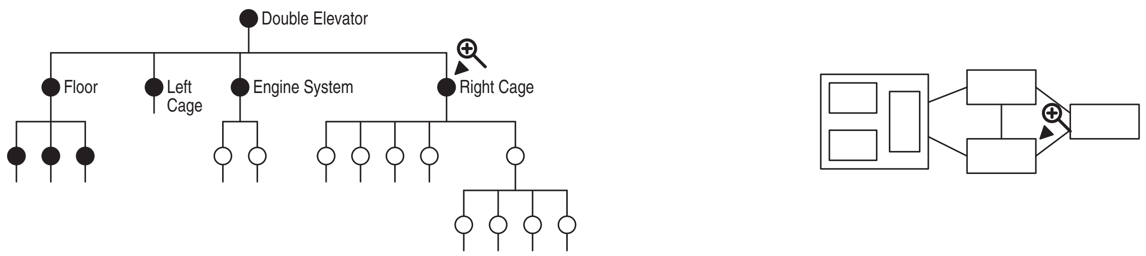

FIGURE 4. Zoom-in request on the Right Cage

Figure 5 shows the view after zoom-ins on the Right Cage composition and then on the Control Panel composition. Right Cage is shown with its successor components. Control Panel also has been exploded and its components Button, Fan Switch, Reservation Keyswitch and Security Keyswitch are now visible. Now a final zoom-in request on Engine System will be performed.

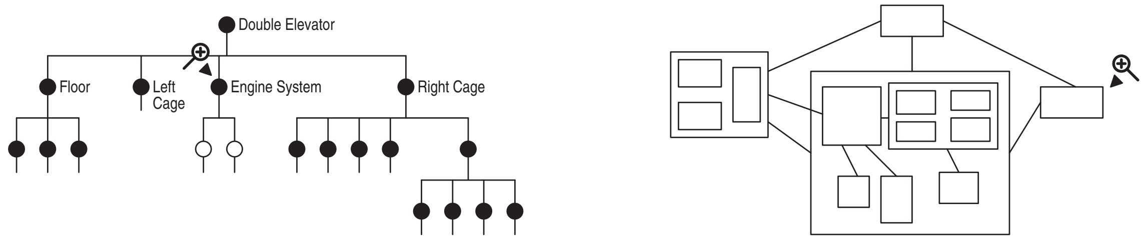

FIGURE 5. Zoom-in request on the Engine System

In figure 6 (see below) all objects are visible. The same components as in figure 1 are visible. To illustrate a zoom-out step, the Right Cage composition will be imploded (see figure 7).

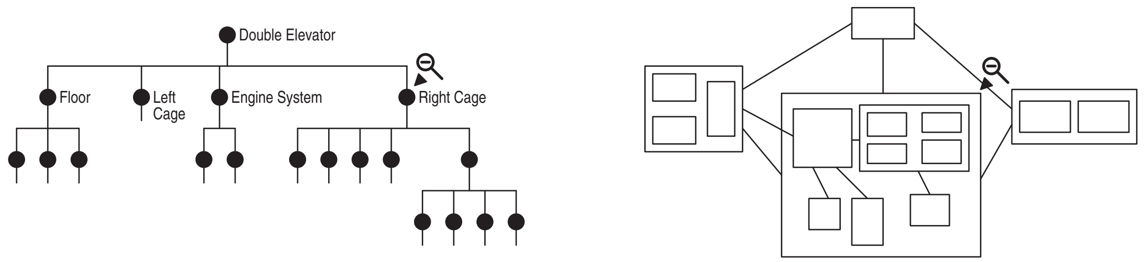

FIGURE 6. Zoom-out request on Right Cage

Figure 7 shows the view after the zoom-out step on the Right Cage composition has been performed. Right Cage has been imploded and the view shown is again similar to figure 4.

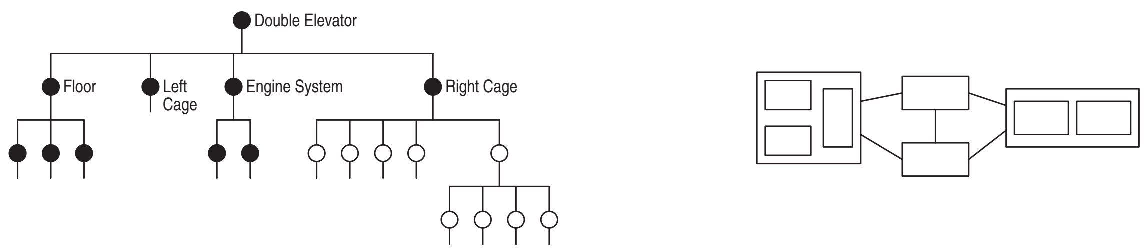

FIGURE 7. View after the zoom-out request

Zooming will be further discussed in the next section (section 5), where the zooming algorithm will be explained in detail.

5 The Zooming Algorithm

In principle the algorithm resizes just the zoomed object and shifts objects away from this object as necessary. When a composition has to be exploded, all objects on the same level will be shifted away from this composition. If necessary the enclosing composition will be resized and all other objects on this level will be shifted. Then the enclosing composition of the enclosing composition will be resized, and so on.

The algorithm operates on any existing layout, adjusting it incrementally. This means that if the user chooses to re-arrange things somewhat, and then expands something, his or her overall rearrangement remains mostly preserved, modulo the incremental repositioning the algorithm performs (a characteristic which is far harder to achieve with an approach that tries to completely automate the entire layout process).

We found that existing fisheye view algorithms were not suitable for visualizing Adora models, because they exclusively provide the automatic generation of a view but do not foresee interactive modification and re-arrangements, respectively. This led us to the algorithmic approach outlined above which is based on a shift of displayed elements on demand. Our realization offers a convenient and simple way to show local detail and global context in one view. It also provides multiple foci and scrolling does not distort the view. Further, we do not need an explicit degree-of-interest-function, which is cumbersome to understand and implement when an arbitrary number of foci must be allowed.

Now we will introduce the algorithm in detail.

Given the center point of a rectangle other than the rectangle to be expanded, figure out which of the four quadrants $P'_1P'_2$, $P'_2P'_3$, $P'_3P'_4$, $P'_4P'_1$ (see figure 9) that center falls in. The quadrant boundaries are lines through the expanded rectangle's opposite corners. If the center lies e.g. in quadrant $P'_4P'_1$, move the rectangle to the right by the amount of the expansion of the expanded rectangle's right side, and upwards by an amount proportional to the degree to which the center point is above the center point (if the center lies in another quadrant apply analogously).

So a rectangle whose center point happened to fall on one of those boundaries would get moved both horizontally by the full horizontal expansion amount, and vertically again by the full vertical expansion amount. This has the result that rectangles that were non-overlapping to begin with, remain non-overlapping.

Based upon the elevator example, figure 8 shows schematically how the algorithm works. First, the leftmost composition (Floor) is zoomed (see figure 8.1). To perform the operation, the new size of the composition in its exploded state is calculated. Then the shift vectors (see A, B, C in figure 8.1) are calculated. These vectors specify the direction and the amount to shift the other three objects, which are on the same level as the composition to be zoomed. All objects which are below the level of the composition to be zoomed and are not components of this composition, must be components of other compositions, and therefore do not need to be explicitly shifted because (see figure 8.2) they will be shifted and redrawn

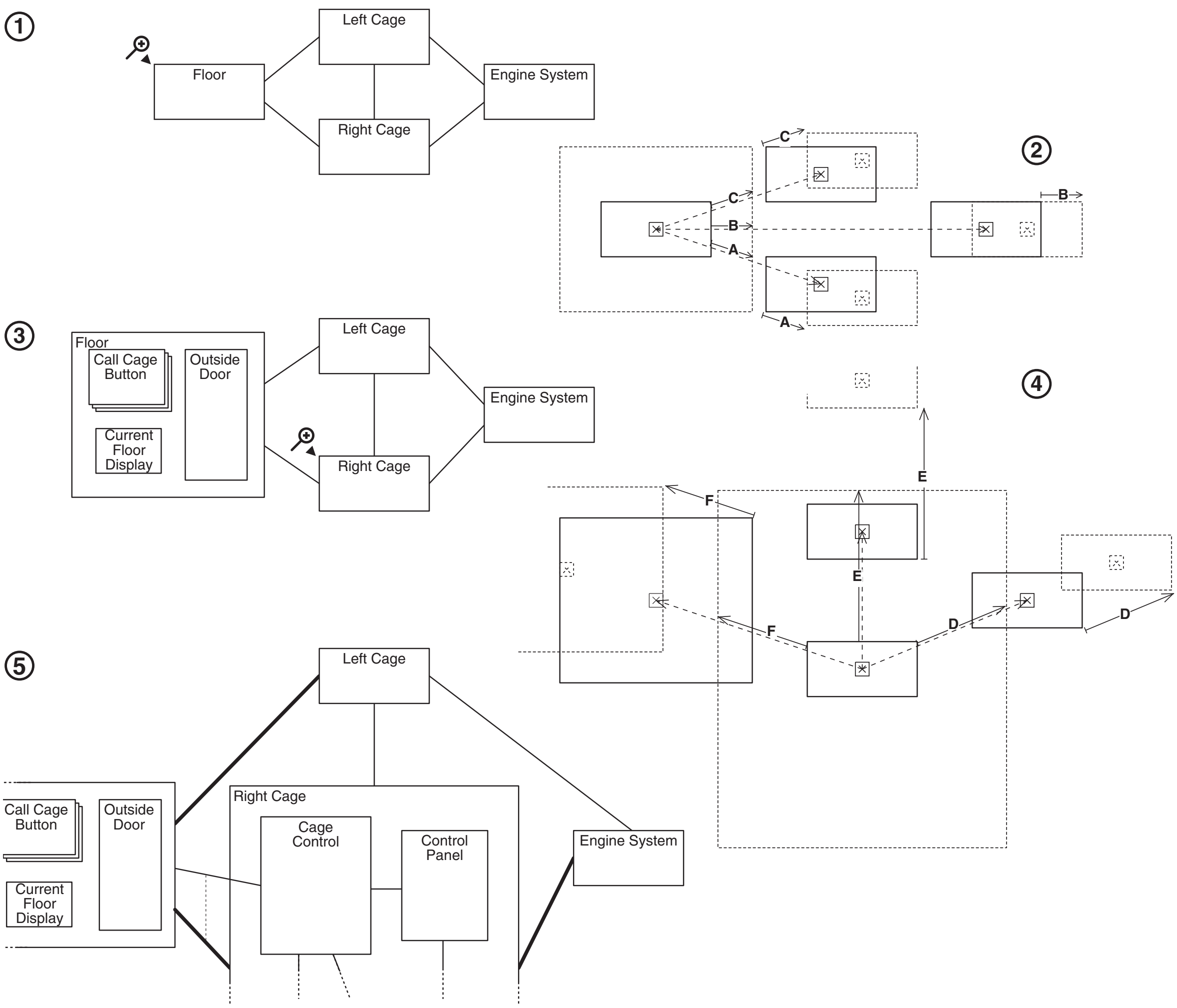

FIGURE 8. Scheme of the zoom-in/reshape algorithm

as their enclosing composition is redrawn. Objects above the level of the zoomed composition will be reshaped as necessary. Figure 8.3 shows the view after the Floor composition has been zoomed. Figure 8.3 and 8.4 illustrate a second zooming step. The following pseudocode sketches how a zoom step is performed:

```
procedure reshape( anObject : Object )
begin
    calculate new size of anObject;
    for each otherObject on the same level as anObject
        calculate shift_vector between anObject and otherObject;
        shift otherObject by the shift_vector
    hcae rof;
    if surroundingCompositionOf( anObject ) must also be reshaped then
        reshape( surroundingComposition( anObject ) )
    fi
end reshape
```

A lot of functionality is hidden behind the calculation of the shift vector $\vec{V}$, which has to be calculated for each component to be moved. The shift vector is calculated in two main steps as follows:

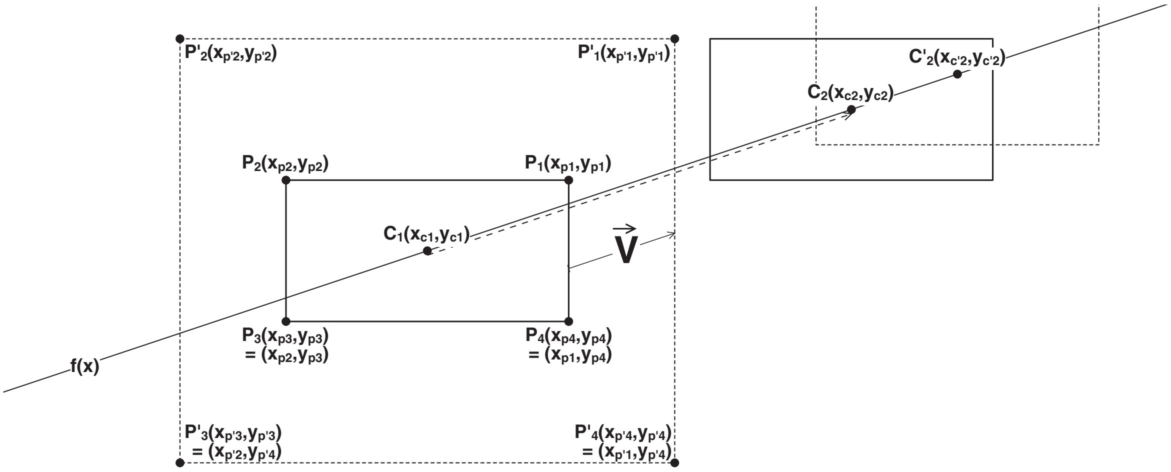

FIGURE 9. Illustration of the data necessary to calculate the shift vector $\vec{V}$

First step: calculate *m* and *a* for the function *f(x)*, which is used in the next step to calculate the shift vector for each object.

$$f(x) = m \cdot x + a$$

$$\text{with} \qquad m = \frac{\Delta y}{\Delta x} = \frac{y_{c_2} - y_{c_1}}{x_{c_2} - x_{c_1}} \qquad \Delta x \neq 0$$

$$\text{and} \qquad a = y_{c_1} - m \cdot x_{c_1}$$

Second step: calculate the shift vector. The conditions are mutually exclusive and determine in which quadrant the center of the object to be shifted falls in. Dependent of the quadrant the center falls in the concrete values for the shift vector can be calculated.

$$\text{to calculate} \qquad \vec{V} = (x_v, y_v)$$

$$\vec{V} = \begin{cases} \text{if } (x_{c_1} < x_{c_2}) \wedge (y_{p'_4} < f(x_{p'_1}) \leq y_{p'_1}) & \text{then} \quad \begin{aligned} x_v &= x_{p'_1} - x_{p_1} \\ y_v &= f(x_{p'_1}) - f(x_{p_1}) \end{aligned} \\ \text{if } (y_{c_1} < y_{c_2}) \wedge (x_{p'_2} \leq f^{-1}(y_{p'_1}) < x_{p'_1}) & \text{then} \quad \begin{aligned} x_v &= f^{-1}(y_{p'_1}) - f^{-1}(y_{p_1}) \\ y_v &= y_{p'_1} - y_{p_1} \end{aligned} \\ \text{if } (x_{c_1} > x_{c_2}) \wedge (y_{p'_3} < f(x_{p'_2}) \leq y_{p'_2}) & \text{then} \quad \begin{aligned} x_v &= x_{p'_2} - x_{p_2} \\ y_v &= f(x_{p'_2}) - f(x_{p_2}) \end{aligned} \\ \text{if } (y_{c_1} > y_{c_2}) \wedge (x_{p'_3} \leq f^{-1}(y_{p'_3}) < x_{p'_4}) & \text{then} \quad \begin{aligned} x_v &= f^{-1}(y_{p'_3}) - f^{-1}(y_{p_3}) \\ y_v &= y_{p'_3} - y_{p_3} \end{aligned} \end{cases}$$

Please note that this is a simplified variant of the algorithm where the shape of an object in its imploded and exploded state must be (similar) rectangles. If the shapes are allowed to be different, the algorithm stays the same in principle, but to determine the shift vector some additional functions are needed to handle a couple of additional conditions. The algorithm has been generalized to handle non-similar rectangles or different shapes but for sake of brevity it will not be described in this paper.

6 Conclusions, State of Work

In this paper we have identified selective zooming as a capability that is useful for, but is lacking in, CASE tools that graphically portray hierarchical structures. We have presented a general concept which transfers the idea of fisheye views to display object hierarchies in an adequate way. We applied this concept to visualize Adora models and demonstrated a possible realization.

However, our visualization concept can also be used for the visualization of other hierarchical (object) models, for example Statecharts [Hare87] or ROOM models [Sel+94]. Moreover, it should be adaptable to the visualization of inheritance hierarchies in object-oriented models, too.

For now, the development of the basic visualization concept for Adora models has been finished, and a first brief validation of this concept has been promising. Currently, we are working on the details and are implementing a graphical model editor, in order to thoroughly validate the usability of the visualization concept. This validation will be done in the Usability Lab of the FIDES Informatik in Zurich. For practical reasons the validation will be done together with the validation of the Adora language.

Based on the results of the validation, our visualization concept will be revised and reimplemented. Furthermore, we plan to build an animator/simulator for Adora models using the same concept.

References

[Aren96] Arena® Simulation-Tool: Systems Modeling Corporation, Sewickely PA, http://www.sm.com/simcontrol.htm; Nov. 1996.

[Bea+90] Beard, D. V., Walker II, J. Q.: Navigational techniques to improve the display of large two-dimensional spaces. *Behaviour & Information Technology*, Vol. 9, No. 6; 1990. (pp. 451-466)

[Booc94] Booch, G.: *Object-Oriented Analysis and Design with Applications*. The Benjamin/Cummings Publishing Company, Inc.; 1994.

[Dart87] Dart, S. A. et al.: Software Development Environments. *IEEE Computer*, Vol. 20, No. 11; Nov. 1987. (pp.18-28)

[Cha+93] Champeaux de, D., D. Lea, P. Faure: *Object-Oriented System Development*. Addison-Wesley Publishing Company, 1993.

[Fer+94] Ferstl, O. K., E. Sinz: *SOM Version 2.10*. Univ. Bamberg, Lehrst. für Wirtschaftsinformatik; 1994.

[Fugg93] Fuggetta, A.: A Classification of CASE Technology. *IEEE Computer*, Vol. 26, No. 12; Dec. 1993. (pp. 25-38)

[Furn86] Furnas, G. W.: Generalized fisheye views. *Proceedings of ACM CHI 86 Conf. on Human Factors in Computing Systems*, Boston, Mass., Apr. 13-17, ACM Press, New York; 1986. (pp. 16-23)

[Gas+95] Gaskell, C., R. Phillips: Software Architecture of the Executable Graphical Specification Tool EGS1. *Software-Concepts and Tools* (1995) 16. (pp. 124-135)

[Glin93] Glinz, M.: Hierarchische Verhaltensbeschreibung in objektorientierten Systemmodellen – eine Grundlage für modellbasiertes Prototyping. In Züllighofen (ed.) / Altmann (coed.) / Doberkat (coed.), *Requirements Engineering 1993: Prototyping*, Bonn, 1993, Teubner Stuttgart; 1993. (pp. 175-192) (hierarchical description of behavior in object-oriented system models)

[Glin95] Glinz, M.: An Integrated Formal Model of Scenarios Based on Statecharts. In Schäfer, W. and Botella, P. (eds.) (1995). *Software Engineering - ESEC '95. Proc. of the 5th Europ. Software Eng. Conf.*, Sitges, Spain. Berlin, etc.: Springer (Lecture Notes in Comp. Sc. 989). (pp. 254-271)

[Hare87] Harel, D.: Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programing*, Vol. 8, No. 3; Jun. 1987. (pp. 231-274)

[Hare88] Harel, D.: On Visual Formalisms. *Comm. of the ACM*, Vol. 31, No. 5; May 1988. (pp. 514-530)

[Inno96] Innovator® CASE Workbench: MID GmbH, Nürnberg, Germany; 1996.

[Jac+94] Jacobson, I., M. Christerson, P. Jonsson, G. Övergaard: *Object-Oriented Software Engineering - A Use Case Driven Approach*. Addison-Wesley Publishing Company; 1994.

[Joo+97] Joos, S., S. Berner, M. Glinz: Hierarchische Zerlegung in objektorientierten Spezifikations-modellen. *Softwaretechnik-Trends*, 17, 1 (Feb. 1997). (pp. 29-37) (hierarchical decomposition in object-oriented specification models)

[Kaen96] Von Känel, R.: *Visualisierungskonzepte von Teil-Ganzes-Hierarchien in objektorientierten Spezifikationsmodellen*. Diploma-Thesis at the Dept. of Comp. Science, Univ. of Zurich; 1996. (visualization concepts for part-of-hierarchies in object-oriented specification models)

[Kel+95] Keller, R. K. et al.: Environment Support for Business Reengineering: The Macrotec Approach. *Software-Concepts and Tools* (1995) 16. (pp. 31-40)

[Koe+96] Kölzer, A., I. Uhe: *Benutzerhandbuch für das KOGGE-Tool BONSAI*. Interner Arbeitsbericht 4/96 der Univ. Konstanz, Inst. für Softwaretechnik; 1996. http://www.uni-koblenz.de/~ist/p_kogge.html; Nov. 1996. (user manual for the KOGGE-tool BONSAI)

[Micr96] MicroSaint® Simulation-Tool: Micro Analysis & Design Inc., Boulder Colorado, http://www.madboulder.com/; Nov. 1996.

[Objy96] Objectory® products: Rational Software Corporation, Santa Clara CA, http://www.rational.com/pst/products/obj_overview.html, Nov. 1996.

[ObjT96] ObjecTime® Toolset: ObjectTime Limited, Kanata, Ontario/Canada, http://www.objectime.on.ca/ObjecTimeProduct.html; Nov. 1996.

[OML96] Firesmith, D., B. H. Henderson-Sellers, I. Graham, M. Page-Jones: *Open Modeling Language (OML) – Reference Manual*. OPEN Consortium; 1996.

[Rose96] Rational Rose® tools: Rational Software Corporation, Santa Clara CA, http://www.rational.com/pst/products/rosefamily.html; Nov. 1996.

[Rum+91] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, W. Lorensen: *Object-Oriented Modeling and Design*. Englewood Cliffs, N. J.: Prentice Hall; 1991.

[Sar+92] Sarkar, M., M. H. Brown: Graphical Fisheye Views of Graphs. *Proceedings of ACM CHI 92 Conference on Human Factors in Computing Systems*, ACM Press, New York; 1992. (pp. 83-91)

[Sch+96] Schaffer, D., et al.: Navigating Hierarchically Clustered Networks through Fisheye and Full-Zoom Methods. *ACM Trans. on Computer-Human Interaction*, Vol. 3, No. 2; Jun. 1996. (pp. 162-188)

[Sel+94] Selic, B., G. Gullekson, P. T. Ward: *Real-Time Object-Oriented Modelling*. John Wiley & Sons, Inc; 1994.

[Shl+88] Shlaer, S., S. J. Mellor: *Object-Oriented Systems Analysis: Modelling the World in Data*. Prentice Hall International, Inc.; 1988.

[Stat96] Statemate® statechart tool: i-Logix Inc., Andover MA, http://www.ilogix.com/; Nov. 1996.

[StP96] Software through Pictures® tools: Interactive Development Environments Inc. / AONIX, SanFrancisco CA, http://www.ide.com/; Nov. 1996.

[SysS96] SystemSpecs® petrinet-Tool: TnTech AG, CH-3018 Bern, http://www.thenet.ch/tntech/sysspecs.html; Nov. 1996.

[UML97] Booch, G., I. Jacobson, J. Rumbaugh: *The Unified Modeling Language for Object-Oriented Development*, Documentation Set Version 1.1, Rational Software Corporation; 1997.

[Wir+90] Wirfs-Brock, R., B. Wilkerson, L. Wiener: *Designing Object-Oriented Software*. Eaglewood Cliffs, N. J.: Prentice Hall Inc.; 1990.