A Visualization Concept for Hierarchical Object Models

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

Martin Arnold
EDS/FIDES Informatik
Zurich, Switzerland
martin.arnold@fides.ch

Abstract

Most current object modeling methods and tools have weaknesses both in the concepts of hierarchical decomposition and in the visualization of these hierarchies. Some methods do not support hierarchical decomposition 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 based on the notion of fisheye views. This concept integrates local detail and global context of a view in the same diagram and eases navigation in hierarchical structures without offending the principle of abstraction. The work presented here is part of an effort to create a method and language called ADORA that provides strong support for hierarchical decomposition.

1 Introduction

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. 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, petri nets, entity-relationship models and nowadays object-oriented models. The purposes of these models range from simple graphic visualization up to achieving specific effects, for example reduction of complexity.

1.1 Model visualization

Using 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 a point of interest (focus) or to move it. The mechanical aspect relates to the mechanical effort (e.g. mouse movement) to achieve a cognitive navigation goal. The cognitive, non-mechanical effort for navigational activities is called *cognitive overhead* [1].

A good visualization concept is critical both for understandability and ease-of-use of graphical models. A good concept should: (1) support *orientation* in the model by visualizing as much local detail as needed without losing the global context of the focused elements, (2) minimize the *cognitive overhead* for navigation, (3) increase *expressiveness* by including the semantics of the model in the visualization, and foster its 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 several tools [2]. Most tools operating on flat or practically flat models only support scaling as a means of handling large models. Few tools have map windows or roam bars [1] for orientation and navigation. 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. Some tools of this kind have scaling possibilities, map windows or hierarchy-overviews to manage the complexity of big models. Most tools provide just *explosive zooming*. As a consequence, these tools offer either views which show *global context without local detail* or *local detail without global context*. 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.

1.2 Motivation and basic ideas

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 supports such a mechanism[1], we are developing a language called ADORA[2] which does. Our work on visualization is based on this language, but our concepts apply to the visualization of hierarchical structures in any graphical 'box-and-line' language.

The principal idea of our approach is to apply the notion of fisheye views [4][8] to the visualization of object 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 rather than being 'just' a geometric projection. 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 elements of special interest, we have strong capabilities for supporting the abstraction mechanisms in the object model that is being visualized and thus foster the expressiveness and understandability of the model.

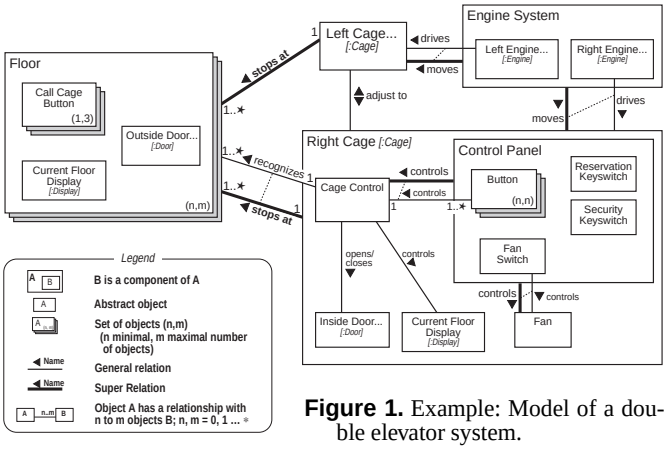

Figure 1. Example: Model of a double elevator system.

2 A brief overview of the ADORA[2] language

ADORA is a semiformal, object-oriented method and language for modeling requirements and software architecture which is currently being developed at the University of Zurich. In this section, we sketch the basic ADORA concepts, in particular, the hierarchical structure of its 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 abstract objects in specific roles instead of classes. Thus we resolve modeling anomalies that occur in decompositions of class models [7].

Hierarchical decomposition and part-of-abstraction are key features 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. Structural relationships and 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 mechanism to objects [5][6].

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 dots following the name denote it to be a composition). (2) Left Cage and Right Cage are different objects of the same type, having different roles (this is an advantage over class models, e.g. UML [11], where this situation cannot be modeled adequately). 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: e.g. between Left Cage and Floor there is only the abstract relationship stops at visible, because Left Cage is displayed without its components. For Right Cage, there is also the component relationship recognizes 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.

3 Navigation in hierarchical models

When we visualize hierarchically structured models, we differentiate between two distinct types of navigation, a physical and a logical one.

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 [1] 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 we want to visualize hierarchical models, we must also be able to navigate through the hierarchy. *Logical navigation* in a hierarchical structure means finding the actual position of a local element in the global context of the hierarchy, or changing the foci of visualized elements. To handle this kind 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 is produced. With *explosive zooming*, the global context gets lost, while the zoomed node explodes entirely in the existing or a new window with all its direct successors. *Fisheye zooming* produces a local detail view while preserving the global context in the same view. The idea is to show local detail – the objects of interest to the

[1] Most existing object-oriented modeling languages either do not fully support a hierarchical model decomposition which employs a *part-of-abstraction* at all [10] or they provide only a clustering mechanism with weak semantics [11]. Few approaches do provide such a decomposition mechanism [3][12]. However, they encounter modeling anomalies which are due to using class models [7].

[2] ADORA stands for *Analysis and Description Of Requirements and Architecture*.

user – in full, while displaying successively less detail for information being further away from this focus.

4 Visualization concept

The tool environment for ADORA (which we are currently developing) consists of a repository, a graphical model editor and an animator/simulator. The general architecture is conventional and straightforward: All functional tool components are grouped around a repository. We will not discuss the general architecture here and concentrate on the visualization concept only.

We analyzed the intended use of the tool and set up navigational scenarios. The three most important ones are: (S1) The user wants to select one, many or all compositions to be displayed in more detail, which means their structure has to be exploded. (S2) The user wants to select one, many or all exploded compositions to be displayed in less detail, which means hiding/abstracting from their internal structure. (S3) The user wants to change the size of the visualized elements without changing the logic structure of the current view. This means shrinking or growing all elements.

From these scenarios, we derived core requirements for our visualization concept, for example: freely selectable degree of detail, local detail and global context in one view, no overlap of objects, multiple foci. Details are presented in [2].

4.1 Concepts

Our basic idea is to use a model-driven fisheye view mechanism with multiple foci to generate views according to the decomposition structure of the model being visualized. Multiple foci means that we can have multiple regions in the model that are displayed in detail.

Navigation. Logical navigation based on explosive zooming does not meet our requirements, because the global context of an object gets lost when zooming-in on that object.

In our concept, a view always represents the *whole* model, but with varying degree of detail. For any object in the view, we can freely select the degree of detail to be visualized, independently of the other objects. Thus the global context of a model is always visible. Details are shown or hidden according to the current interest of the viewer. For logical navigation, we use *selective* zooming. We zoom-in on objects of interest and zoom-out on the others. Additionally, we provide conventional *scaling* and *scrolling* navigation for views which are larger than the display screen. This is done in the usual way and is not discussed here.

Selective 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, internal structure of this composition.

In any visualized configuration, each composition can be zoomed-in or zoomed-out independently of the visualized context surrounding this composition. With this idea we realize *multiple foci.*

4.2 Example

Starting point for the illustration of the zooming concept is the situation presented in Fig. 2: The double elevator system (Fig. 1) 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 in the tree on the left hand side of each figure by filled black circles. The example will illustrate a sequence of zooms. A magnifying glasses indicates a composition to be zoomed.

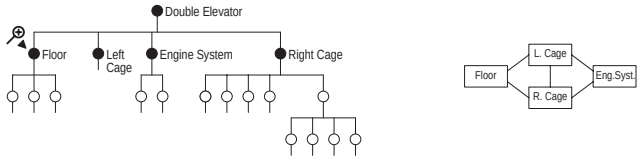

Figure 2. Initial view, new zoom-in request on Floor.

In the example, we proceed as follows: We zoom-in until all objects of the hierarchy are visible (same view as in Fig. 1) and then perform one zoom-out. The first step is a zoom-in request on the Floor composition. The Floor composition will be exploded and its internal structure becomes visible. Fig. 3 shows the resulting view. Floor is displayed with its direct successor components Call Cage Button, Current Floor Display and Outside Door.

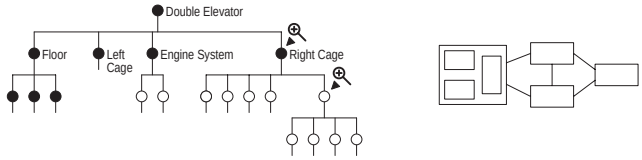

Figure 3. View after Floor has been exploded; new zoom-in requests on Right Cage and Control Panel.

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. Fig. 4 shows the view after a zoom-in on the Right Cage and the Control Panel composition. Right Cage is shown with its components. Control Panel has also been exploded and its components Button, Fan Switch, Reservation Keyswitch and Security Keyswitch are now visible. Now a final

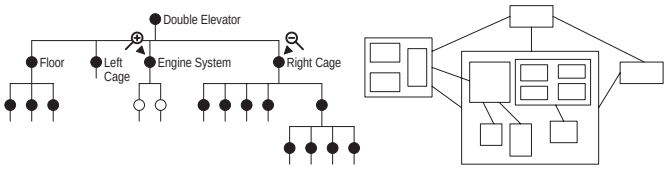

Figure 4. View after Right Cage and Control Panel have been exploded; new zoom-in request on Engine System and zoom-out request on Right Cage.

zoom-in request on Engine System will be performed to have all objects visible (similar to Fig. 1). Thereafter, a zoom-out request on Right Cage will be performed (see Fig. 5).

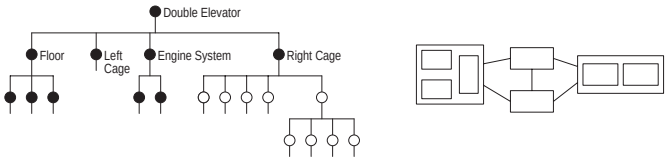

Figure 5. View after Engine System has been exploded and Right Cage has been imploded.

4.3 The zooming-algorithm

Below we sketch the algorithm for zooming-in. A detailed description is given in [2]. The algorithm consists of three main steps: (1) Calculate the new size of the object that is zoomed-in (depending on the number of details to be shown). (2) For each surrounding object x on the same hierarchical level, calculate a vector V_x and shift the object by this vector (Fig. 6). (3) If surrounding compositions have to be reshaped, reshape them recursively (inside-out), using steps (1) and (2).

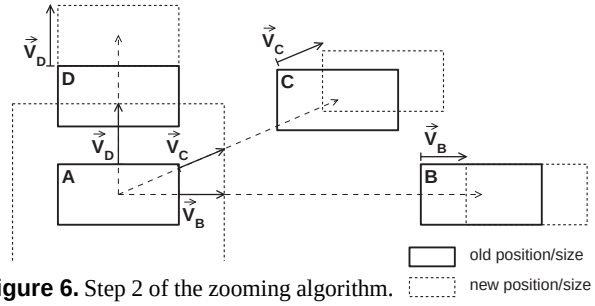

Figure 6. Step 2 of the zooming algorithm.

The algorithm works on any given layout, adjusting it incrementally and preserving it as far as possible. So a user may re-arrange a layout without losing these rearrangements when zooming. This is a distinct feature of our approach. Existing fisheye view algorithms use automatic layout generation and do not permit users to re-arrange the layout of the objects being displayed.

5 Conclusions, state of work

In this paper we have identified selective, fisheye-style 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 that transfers the idea of fisheye views to display object hierarchies in an adequate way. We applied this concept for the visualization of ADORA models and demonstrated a possible realization in [2].

However, our visualization concept can also be used for the visualization of other hierarchical models, for example statecharts or ROOM models [9]. Moreover, it should be adaptable to the visualization of inheritance hierarchies in object-oriented models, too.

The development of the basic visualization concept for ADORA models is finished, and a first brief validation has been promising. Currently, we are working on the details and are building a prototype model editor which implements our visualization concept. The usability of selective zooming will be thoroughly validated by testing our prototype in the Usability Lab of EDS/FIDES Informatik in Zurich. Based on the results of this evaluation, we plan to revise and reimplement our concept. In parallel, we are continuing the development of the ADORA method and language.

6 References

[1] Beard, D. V., Walker II, J. Q.: Navigational techniques to improve the display of large 2-D spaces. *Behaviour & Information Technology*, Vol. 9, No. 6; 1990. (pp. 451-466)

[2] Berner, S, S. Joos, M. Glinz, M. Arnold: Visualizing Adora Models. TR-98-09, Department of Computer Science, University of Zurich, 1998.

[3] Champeaux de, D., D. Lea, P. Faure: *Object-Oriented System Development*. Addison-Wesley, 1993.

[4] Furnas, G. W.: Generalized fisheye views. *Proc. of ACM CHI 86 Conference on Human Factors in Computing Systems.* Boston, Mass., ACM Press, New York; 1986. (pp. 16-23)

[5] Glinz, M.: Hierarchische Verhaltensbeschreibung in objektorientierten Systemmodellen. In Züllighofen, Altmann, Doberkat (eds.). *Requirements Engineering 1993: Prototyping*. Stuttgart: Teubner; 1993. (pp. 175-192) [Hierarchical description of behavior in object-oriented system models; in German]

[6] Glinz, M.: An Integrated Formal Model of Scenarios Based on Statecharts. In Schäfer, Botella (eds.). *Software Engineering - ESEC '95. Proc. of the 5th ESEC*. Sitges, Spain. Berlin, etc.: Springer; 1995. (pp. 254-271)

[7] Joos, S., S. Berner, M. Glinz: Hierarchische Zerlegung in objektorientierten Spezifikationsmodellen. *Softwaretechnik-Trends*, Vol 7, No. 1; Feb 1997. (pp. 29-37) [Hierarchical decomposition in object-oriented specification models; in German]

[8] Schaffer, D., et al.: Navigating Hierarchically Clustered Networks through Fisheye and Full-Zoom Methods. *ACM Transactions on CHI*, Vol. 3, No. 2; Jun. 1996. (pp. 162-188)

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

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

[11] Booch, G., I. Jacobson, J. Rumbaugh: *The Unified Modeling Language for Object-Oriented Development*, Documentation Set v1.1, Rational Software Corp.; 1997.

[12] Wirfs-Brock, R., B. Wilkerson, L. Wiener: *Designing Object-Oriented Software*. Prentice Hall; 1990.