The ZOOM ILLUSTRATOR as a MMPS

Bernhard Preim

Institut für Simulation und Graphik, Otto-von-Guericke-Universität Magdeburg
Universitätsplatz 2, D-39106 Magdeburg
e-mail: bernhard@isg.cs.uni-magdeburg.de

Abstract. We describe the architecture of the ZOOM ILLUSTRATOR – an interactive system to illustrate spatial phenomena. The system focuses on the combination of rendered images with textual descriptions. Fisheye techniques are employed to present detailed information while maintaining the context of the overall available information.

Our system was designed in an object-oriented manner. In this paper, we focus on an OO-description of the ZOOM ILLUSTRATOR compared to a system description in terms of the SRM. The feasibility of a description in terms of the SRM encourages its use for the description of interactive systems.

Our conclusion is that the OO- and the SRM-description complement one another. In unison with one another, they lead to a significantly more structured system description than could be attained with either of the methods taken by itself.

Keywords: Interactive Illustrations, Image-Text-Relation, Fisheye techniques, Multimodal Presentation, Object-Oriented Design

1 INTRODUCTION

Interactive 3D graphics offers high potential for the explanation of spatial phenomena, as can be found for example in engineering and anatomy. The interactive handling of 3D models is important to clarify spatial relations. While this is well-recognized, not enough effort has been spent on the combination of rendered images and textual information.

Borrowing from textbooks gives hints on how to combine images and text. Images are often surrounded by labels referring to their parts via reference lines. Explanations refer to the spatial structure and are enhanced by cross references as to spatial relations. In textbooks, however, explanations are generally not integrated in an illustration but are placed under an image or even on a separate page, which complicates comprehension. Interactive systems can handle this problem and tailor the presentation to the information requested.

Based on these observations we developed the ZOOM ILLUSTRATOR. Our system tackles some problems which are typical in multimodal presentation systems, such as:

- The use of different modalities, namely images and text, must be coordinated.
- Flexible layout strategies are required. This is due to the amount of textual information related to a complex 3D model.
- User support is necessary for the annotation of graphical objects with textual labels.

To adapt a presentation to the user's interest, fisheye techniques as introduced by FURNAS (see [3]) are very useful. Fisheye techniques place and scale the space to accommodate information depending on the user's interest and allow the user to look at something in detail while maintaining the context. Important information is emphasized, driven by a *degree-of-interest* (DOI) which is assigned to each piece of information. DOI-values depend on a static à priori importance (API) and on dynamic factors which consider the (spatial or cognitive) distance of pieces of information from the one the user interacts with. DILL *et al.* (see [2]) developed the continuous zoom, a fisheye algorithm, which provides smooth transitions between an original layout and the layout after a zoom operation. As continuous transitions enhance the user's understanding we employ a variant of this algorithm. Fisheye views are exploited to integrate detailed textual information (e.g. explanations) and context (the labels of important objects).

Our system aims to provide interaction facilities for an end-user. However, there is not a strict separation between knowledge-based systems and interactive systems, because the latter can benefit from automatic support to allow high-level interaction.

This paper describes the architecture of an interactive illustration system. In accordance with its purpose, to integrate images and textual descriptions, that is to *illustrate*, and its basic interaction technique, to *zoom*, we refer to our system as ZOOM ILLUSTRATOR.

The system has been developed using the Object-Oriented approach. In this paper, we compare the suitability of the Object Oriented Design (OO-Design) and of the SRM for the description of the system's architecture. It turns out that the SRM is able to capture our design and offers new ways to look at the system.

2 ARCHITECTURE OF THE ZOOM ILLUSTRATOR

With our design, we combine interactive 3D graphics with hypertext functionality for educational purposes. This involves the presentation of more or less detailed information on the textual part and results in a continuous zoom of the corresponding node to accommodate this information, while automatically repositioning and rescaling other nodes. On the graphical side, direct manipulation of a 3D model is offered.

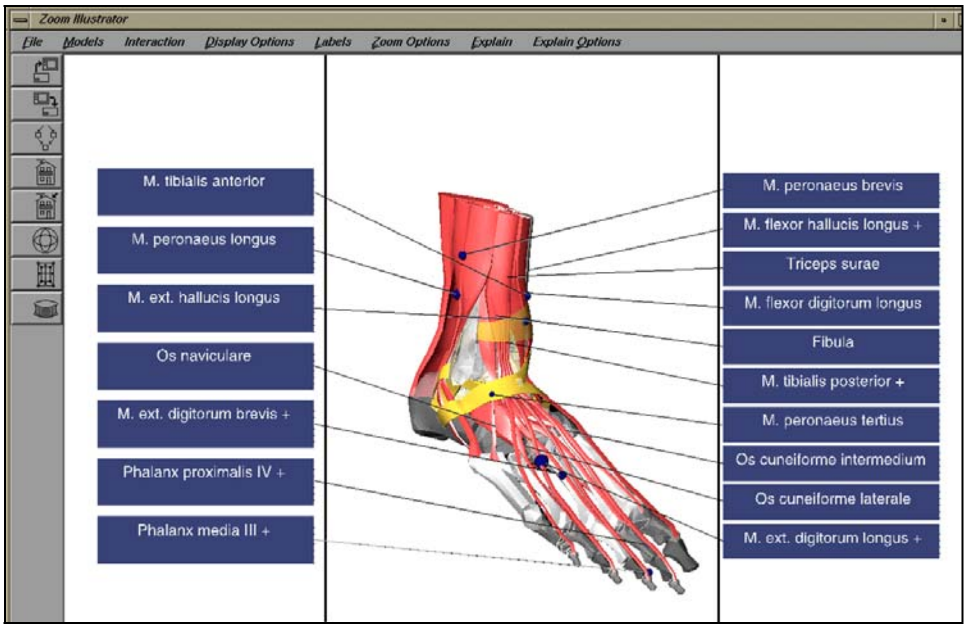


Figure 1: Basic layout of the ZOOM ILLUSTRATOR with lines superimposed to indicate the separation between the graphics- and text area. Reference lines connect the image and the labels with each other. The endpoints of reference lines are marked with small spheres.

The initial layout encompasses the most important labels and one instance of the 3D model to illustrate. The selection of the „most important“ labels is based on the API-values. These values (recall [3]) are derived from the object structure (the position within the hierarchy) and from geometric criteria (size and visibility of objects). The *Layout Manager* is responsible for choosing the most important nodes to label (see the architecture in Figure 2). Textual information is distributed as evenly as possible between the left and the right side of the image. Separate areas for the graphics- and text presentation are necessary to ensure that they do not occlude each other. Furthermore, the *Layout Manager* tries to avoid reference lines which cross each other. Figure 1 shows an output of the system which is generated from the user's specification to see an anatomic model with the focus on the muscles.

Figure 2 summarizes the architecture of the ZOOM ILLUSTRATOR. The generation of an illustration is based on two sources (see the top vertical boxes): The first is a *Scene Description*, containing a polygonal 3D model which is structured into objects. Secondly, we employ a file with related *Textual Descriptions* referring to the objects in the scene description. The *Textual Description* contains labels and explanations. After the sources are loaded, an internal representation is generated using the correspondence between a common key in the *Scene Description* and in the *Textual Description*. A user can interact with the *Text Display* (e.g. ask for an explanation) and with the *Image Display* (e.g. modify individual objects).

The architecture is well-suited to communicate the ideas behind the ZOOM ILLUSTRATOR. It is abstract insofar as it does not lead to an implementation in a straightforward manner. The terminology is dedicated to the basic features and techniques of our specific system.

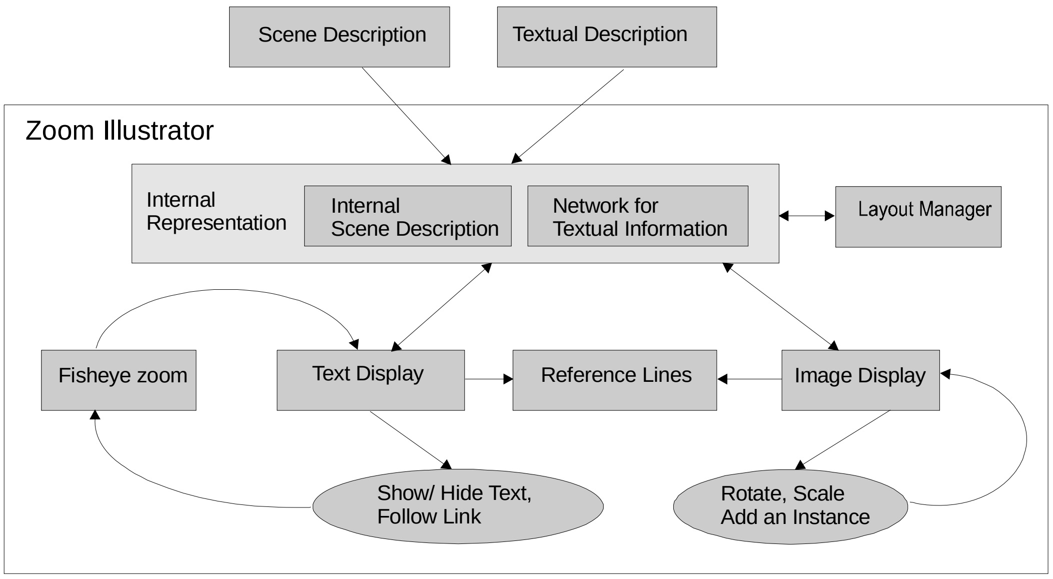


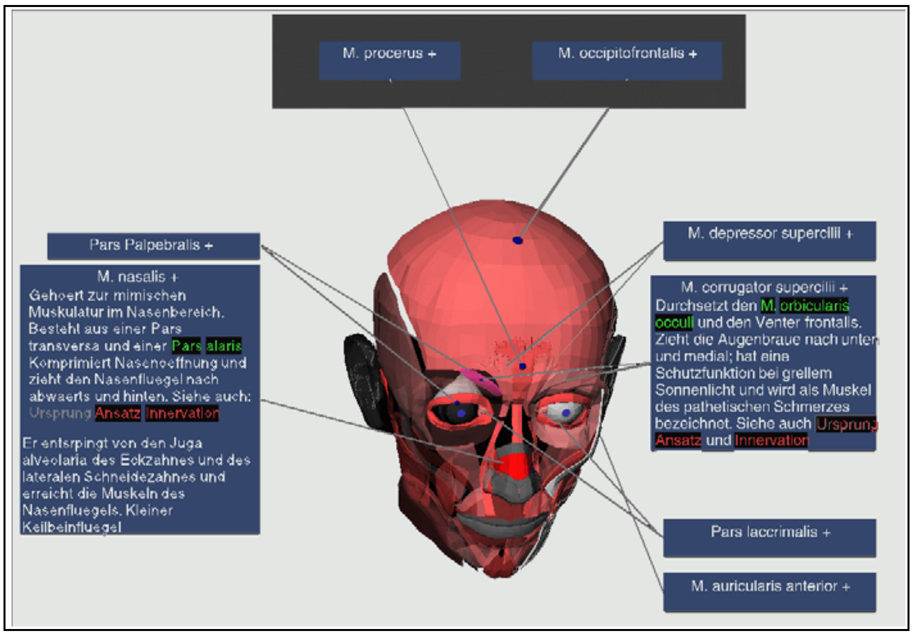
Figure 2: Overall Architecture

Interaction Facilities

Navigation on the textual part, more detailed information can be requested for a node. This request initiates a *zoom operation* (a sequence of small *zoom steps* to realize a size request with „continuous“ changes). to accommodate an explanation (see Figure 3). Furthermore hyperlinks may be followed which connect different explanations for one node as well as explanations of different nodes with each other. As usual, e.g. in Web-Browsers, the colour of hyperlinks is adapted to whether or not they have been visited.

On the graphical part, 3D models can be transformed freely. Furthermore, changes on the textual presentation cause an adaptation of the corresponding graphical part. Transparency and saturation of colours have proven to be important parameters to adapt an object's appearance to the amount of detail of its textual description. Figure 3 shows a typical output of the system which incorporates this adaptation.

Originally, the zoom algorithm placed and scaled the whole space to accommodate textual information. In many cases, the zoom performs well and does what can be expected. However, if one node is zoomed up, others may be closed, which is an unwanted side-effect. To prevent nodes from being closed due to a zoom operation, they can be transferred to the *pinnwall* (see Figure 3). This is a container for a few „privileged“ nodes which are not exposed to the fisheye zoom. These nodes remain at fixed positions and are always connected to the graphics.



M. nasalis +
Gehoert zur mimischen Muskulatur im Nasenbereich. Besteht aus einer Pars transversa und einer Pars alaris Komprimiert Nasenoeffnung und zieht den Nasenfluegel nach abwaerts und hinten. Siehe auch: Ursprung Ansatz Innervation

Er entsrpingt von den Juga alveolaria des Eckzahnes und des lateralen Schneidezahnes und erreicht die Muskeln des Nasenfluegels. Kleiner Keilbeinfluegel

M. corrugator supercilii +
Durchsetzt den M. orbicularis occuli und den Venter frontalis. Zieht die Augenbraue nach unten und medial; hat eine Schutzfunktion bei grellem Sonnenlicht und wird als Muskel des pathetischen Schmerzes bezeichnet. Siehe auch Ursprun Ansatz und Innervation

Figure 3: Example layout with different levels of text presentation ranging from a label to an extended explanation on the left side.

3 Object-Oriented Design

Following the conceptual architecture, this Section describes the OO Design of the Zoom Illustrator. *Classnames*, *methods* and *members* are written in italic. The spelling *Class:: method()* refers to a method of a class. The *has-a* relations between the important classes are shown in Figure 4. If there is only one instance of a *Class* it is referred to as the *Class*.

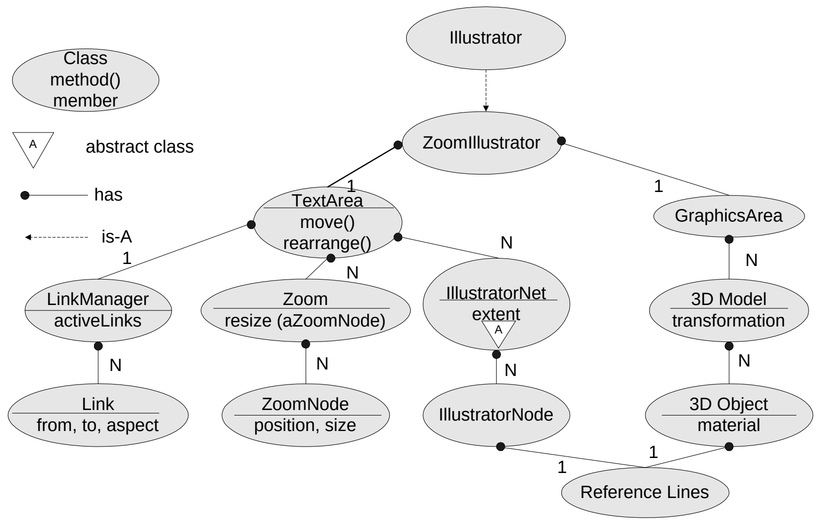


Figure 4: Relations between the most important classes, notation according to Booch (see [1])

Design of the toplevel-classes

The basic idea of the OOD is to regard an illustration as a hierarchy of parts which manage a media or a screen space. This implies that one instance is on top of the hierarchical composition of the illustration. To realize the described architecture, images and text must be managed. This gives rise to the definition of the classes *TextArea* and *GraphicsArea*. These classes manage the extent of their media and contain children, instances of the 3D model and networks of textual information. They assign space to their children, cause a rearrangement of their children (e.g. if a new child is added). Both classes involve some file operations, namely they load and process information from an external file. The work of the *TextArea* and the *GraphicsArea* is coordinated by an *Illustrator*, which is on top of the hierarchical representation. It communicates directly with the application and distributes commands to their children (namely, *GraphicsArea* and *TextArea*). The similarities between these three classes give rise to the definition of a common base class which we call *IllustratorPart* (see Figure 5 with the *is-a* relations). It has the members *children* and *screenspace* and methods to manipulate them. The peculiarities of the part which is on top of the illustration are summarized in a class *TopLevelPart*. This class is derived from *IllustratorPart* and enhances the inherited behavior by methods to manipulate the viewing window in which the illustration is presented and global rendering parameters.

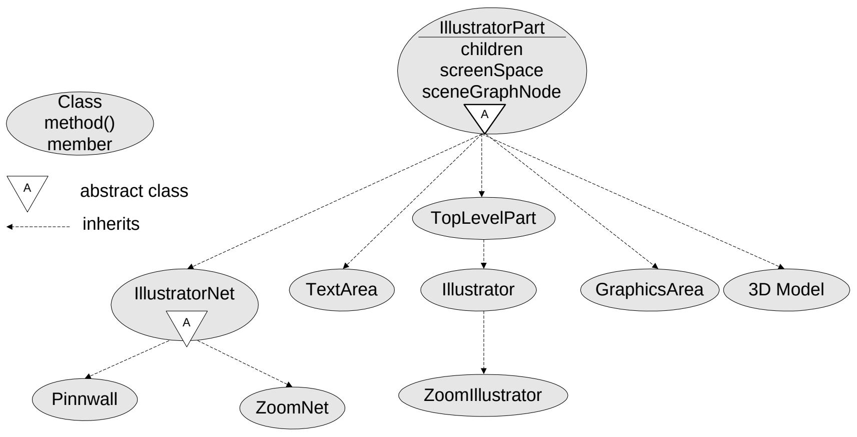


Figure 5: The *is-a* relations between the most important classes

With this policy a strict separation of concerns is achieved. The abstract class *IllustratorPart* provides the basic services for all parts of an illustration, where some methods are purely virtual, requiring reimplementation in derived classes. Class *TopLevelPart* provides the generic behavior of that part which is on top of the illustration. Class *Illustrator* is less generic and concentrates on the coordination of images and text. Furthermore, the *Illustrator* is responsible for the overall layout of the illustration. An even more dedicated class is derived from class *Illustrator* and provides the specific behaviour required to control the fisheye zoom. This derived class is called *ZoomIllustrator*.

Graphical Interaction

The *GraphicsArea* manages *3D Models*. Each *3D Model* instance has its own *transformation* to be rotated and scaled independently. Figure 6 demonstrates the independent handling of two *3D Models*. *3D Models* are containers for *3D Objects*, with each object having its own material information. With this structure the material of each *3D Object* can be changed independently in several instances of the *3D Model*. Thus, an object can be emphasized selectively in one instance of the *3D Model*.

Media Coordination

One important issue in a Multimodal Presentation System is the coordination of the different media. This task is carried out by the *Illustrator*, which distributes commands at the highest level. Coordination at a lower level (between one piece of text and the related graphics part, is carried out by a collaboration between *IllustratorNode* and class *3D Object*.

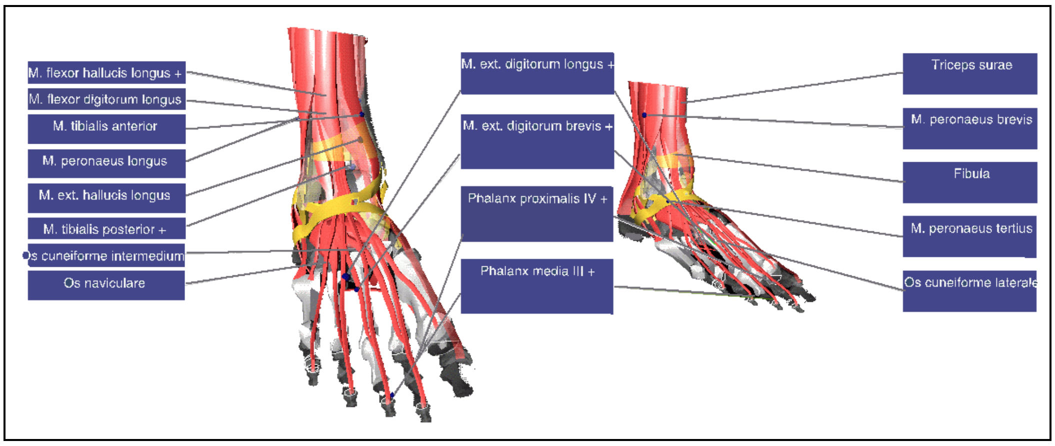


Figure 6: Independent handling of several instances of a 3D model. Labels which refer to objects which are visible in both instances are placed between them and connected to both.

Requests to change the view, to add or remove an instance of the *3D Model* are propagated to the *TextArea* which initiates an update of all textual information and of all reference lines. The *TextArea* manages several *IllustratorNets* which "know" their extent on the screen and hold a list of *IllustratorNodes*. The *IllustratorNets* initiate the update process for their *Illustrator-Nodes*, which includes the adaptation of the selected representation to the space available. Requests to explain a node are processed from the textual part, resulting in a zoom operation to accommodate the desired text, and from the graphics part. On the graphical part the *3D Object* to be explained is emphasized (with a more saturated colour) and those *3D Objects* which may occlude it are deaccentuated performing changes of material properties. If necessary, the *3D Model* is transformed to ensure visibility of the explained object.

For the emphasis of small graphical objects, however, it is not enough to choose an appropriate viewing direction and to emphasize an object with an appropriate colour. In this case, a 3D fisheye zoom is exploited to enlarge the graphical detail at the expense of other objects. The incorporation of the 3D fisheye zoom in the ZOOM ILLUSTRATOR is described in [5]. With the 3D fisheye zoom the navigation in the 3D model and in the textual information is unified.

Navigation in Textual Information

The *LinkManager* maintains all hypertext links currently presented an explanation. This makes it possible to adapt the presentation of nodes (e.g. the label colour) if they are mentioned somewhere else in an explanation. A *Link* is a relation between two nodes under a certain aspect, that is *node*$_1$ has an *aspect* (an explanation) which refers to *node*$_2$. As shown in Figure 4, *IllustratorNet* is an abstract class. Subclasses have to define a layout strategy. Two subclasses are defined: *ZoomNet*, which uses the zoom algorithm for the layout and *PinWallNet*, with nodes at fixed positions. Instances of *ZoomNet* manage the labels on the left and right side whereas a *PinnWallNet* is responsible for the nodes on top of the image. The *Zoom* realizes the continuous zoom algorithm and initiates an update of all *ZoomNodes* after a zoom step.

Controlling Text Layout

Important methods of class *TextArea* are *move()* and *rearrange()*. The *move()*-method transfers a node to another network by an animation. It is invoked for instance to shift a node to the *PinWallNet*. The movement consists of three steps: A zoom operation to provide the space in the target network, a second zoom operation to distribute the space which is no longer needed in the source network and the actual movement from the source network to the target area. The *rearrange()*-method redistributes textual information to prevent crossing reference lines (which may occur when transforming the 3D model). This process is only invoked after an explicit command from the user to avoid confusions resulting from "flying" labels.

Managing Information for one node

An *IllustratorNode* manages application-specific data related to a node. This includes information about its appearance (e.g. colours of the label, of the rectangular area) and about its relation to the graphics part, summarized in an instance of *ReferenceLine*. This instance holds the reference object within the graphic (see the association between an *IllustratorNode* and a *3D Object* in Figure 4) and allows an *IllustratorNode* to adapt the appearance of the graphical counterpart.

All information for the placement of an *IllustratorNode* is encapsulated in the *ZoomNode* (the position, the scale factor as a request to the *Zoom*). The separation between zoom-specific (or more general layout-specific) information and application-specific information (managed by an *IllustratorNode*) has proven to be very useful for modifications.

Representations of different levels of textual description

An important issue in the design of the ZOOM ILLUSTRATOR is the concept of *representations*. Each node has several representations where the set of representations varies depending on the node's category. Each node has at least a label and it may have explanations.

A *Representation* is an abstract class. It is characterized by its *level of detail* and the *neededSize* to be activated (higher levels of detail corresponding to more information and more space needed). Each representation has a *display()*-method to activate itself.

Representations log their activations. This is useful if several representations have the same *level of detail.* In this case a direct mapping of the DOI to a representation is not possible, instead the most suitable should be selected on the base of previous activations. The strategies for the selection of representations are described in detail in Rüger *et al.* (see [6]).

Label and *Explanation* are subclasses of *Representation* which differ in their *display()*-method. The *Label::display()*-method simply decides whether the label string or a shorter string with abbreviations can be accommodated and displays it. *Explanation::display()* arranges a longer text in a rectangle with a given width with bold parts and hyper links to other nodes (*global links*) as well as to other explanations of the same node (*local links*).

The explanations presented, however, are prepared text-sequences and not the result of a natural language generation (NLG) process. The text presentation would benefit from NLG, not only because more flexibility would be possible. A semantic representation behind the text would make it possible to adapt the 3D model to the very specific explanation currently explained. The presentation of a muscle can be adapted according to whether its orign, its shape or function is explained.

Annotating 3D Models

An important issue in generating illustrations which integrate images and text is the annotation of 3D models. As images and text are presented in separate areas, each graphical object must be connected to the corresponding part in the textual information via a line. In particular, reference points within an image have to be calculated to which reference lines point. This calculation should fulfil several requirements:

1. The reference point should clearly belong to the object to be annotated. This is not as trivial as it may seem. The bounding box centre or the centre point, which seem to be good candidates, may be outside the object if it has a concave shape.

2. The resulting point should be visible, which is even less trivial and implies that the calculation is view-point dependent.

3. Finally, it should be fast. This implies that the calculation of reference points for complex objects cannot take into account all surface points of the object.

The first requirements can be fulfilled using a vertex of that object. All vertices obviously fulfil the first requirement, and if none of them is visible it is very likely that the whole object is invisible and the second requirement cannot be fulfilled. To ensure that the calculation is finished in a reasonable amount of time (3), the calculation is adapted to the number of vertices of an object (for objects with many vertices only a fraction is tested as to whether they are appropriate as reference points). The annotation is achieved as a collaboration of an *IllustratorNode* (which holds the coordinates of the label) and the related *3D-Object* which holds the coordinates of the reference points.

4 DESCRIPTION IN TERMS OF THE SRM

Our system targets at interactive behaviour and even those features of the system which may sound "intelligent" at first glance, like the selection of representations, are in fact not. Instead a simple comparison between numerical values takes place. In a system where complex 3D models are interactively transformed, this is necessary just to keep the response rate reasonable. Although the ZOOM ILLUSTRATOR performs some kind of automatic support (annotation, layout, coordination), it is rather a (simple) Multimedia Presentation System than an "intelligent" one.

Despite this difference we attempt to characterize the ZOOM ILLUSTRATOR within the terminology of the SRM. It turns out that this is feasible and (at least for the author) interesting. According to the structure of the SRM this description is divided into two parts: The first describes which components of the SRM exist in our system and the second part summarizes their relations in an architectural scheme which is more general than the one presented in Figure 2.

4.1 Important Terms

Media

The ZOOM ILLUSTRATOR includes written text and graphics which is generated when requested. Output media supported include the screen and printers, where an Offscreen-Renderer generates a postscript-file with the rendered presentation.

Goals

Goals to be achieved include textual explanations (*explain* anObject| aGroup *under* anAspect), viewing from arbitrary directions (*show* aModel *from* direction$_1$ {*and from* direction$_2$}. These goals are not directly specified by a user, instead he or she invokes commands via mouseclicks or menu-items.

The explicit formulation of these goals, however, is a prerequisite for constructing scripts which are interpreted – resulting in commands to the ZOOM ILLUSTRATOR to build up an animation sequence. The extension of the interactive system to a semi-interactive tool, the development of a scripting language and its combination with the interactive kernel is described in [5].

Presentational commands

To enable the user to achieve these goals, interaction facilities on the textual side as well as for the manipulation of the graphics are provided. When mouse-based invocation of commands does not seem to be reasonable, menu items are provided, e.g. to add/remove an instance of the *3D Model*, to rearrange labels after geometric transformations.

Application

The external data sources for our system are the textual description and the scene description (recall Figure 2). The scene description describes the geometry of the underlying 3D model. The textual information consists of two parts:

- Structure information
  ⇒ Existing categories and subcategories
  (e.g. in anatomy muscles, bones and nerves) where subcategories summarize nodes of a category which belong to a certain region (e.g. face muscles)

⇒ Linkage of nodes via hyperlinks

⇒ Nodes and representations which belong to a (sub)category

- Textual Information for each node structured according to the first part.

The separation of structure information and textual information enables us to decide quickly which nodes are important to adapt the presentation in a specific context. Structure information is used to find out which items are important with respect to a specified goal.

Knowledge

The ZOOM ILLUSTRATOR records requests to change the representation of a node. This information is made explicit as a history which contains items of the form (<*node*>, <*representation*>) and furthermore exploited to guide the presentation (recall Section 3). In the terminology of the SRM, this information belongs to the *Discourse Model*. Changes on the graphical part are also recorded and thus the whole illustration can be reconstructed, which makes it possible to return to arbitrary points in the history of interaction.

Design Knowledge

Design knowledge is included in the software, but not explicitly stored in a knowledge-base. The required knowledge can be categorized in knowledge on how to focus on pieces of information, how to relate graphics and text to each other and on how to make changes smooth. Smooth changes are important for both the system (to prevent unnecessary rendering) and even more for the user (to provide animated movements instead of rapid changes).

To make geometric transformations smooth, knowledge about rendering parameters, their influence on the quality of the image and on rendering times is required. The SRM refers to this knowledge as *Media Specific Design Knowledge*. The basic principle is to find a trade-off between quality (essential for looking in detail) and the frame-rate (essential when transforming the 3D model). The incorporation of this knowledge is crucial for the acceptance of the system, however it depends strongly on a specific environment.

4.2 Architecture in terms of the SRM

The description of the ZOOM ILLUSTRATOR in terms of the SRM, reveals that two important parts are not present (see Figure 7). On the part of the layers, the *Content Layer* is missing. A selection of media and of strategies for coordination does not exist. The two basic media are always employed, their coordination is defined by user-defined options and changes in one medium are propagated to the corresponding part in the other medium. Due to the interaction facilities offered and the narrow target area, to illustrate complex spatial phenomena, a planning scheme concerning how to present something is less important than for systems with a broader scope, like WIP (see WAHLSTER *et al.* in [8]).

On the other hand, there is no *User Expert* in the scheme, meaning that information concerning the user is not stored. To tailor the presentation himself or herself, the user is offered *Display Options* (including fonts and colours), and *Rendering Options* (concerning the compromises between quality and frame-rate, forcing different rendering algorithms to apply).

Application Expert

The *Application Expert* contains structure information about the domain, including categories (e.g. muscles), aspects of textual descriptions for nodes of a layer and has knowledge as to which objects are visible from which directions (visibility information derived from a 3D model). The *Application Expert* has knowledge how to present objects of a category graphically (colours and material properties) and from standardized viewing directions.

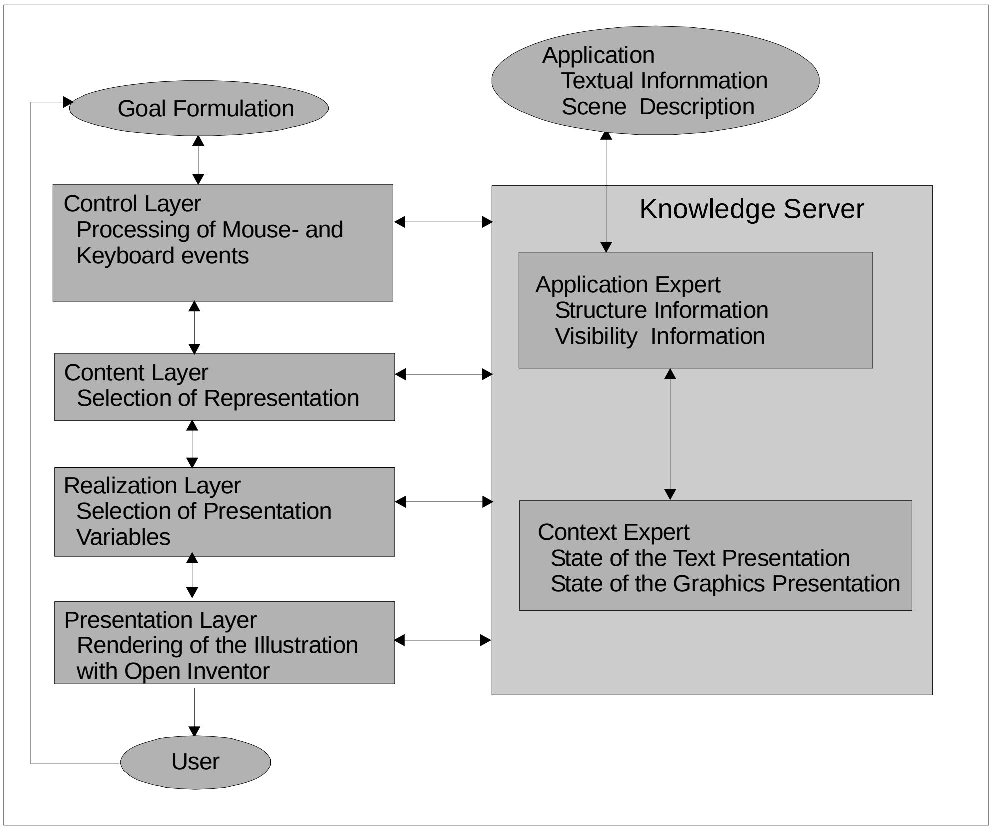


Figure 7: Architecture in terms of the SRM

Context Expert

The design of a *ContextExpert* for our system is strongly influenced by the Reference Architecture. It contains information on active links, on previous activations of textual information, which is exploited to adapt the presentation (e.g. representations to activate, colours). Furthermore information about the state of the graphics is included. For each *3D Model* the transformation and the state of manipulators is registered, which enables us to save a „homePosition“ to which the user may return. For each *3D Object* the *Context Expert* stores its material and the draw-style (wireframe, filled).

The *Context Expert* is „informed“ whenever an event occurs which changes the presentation and stores it into a list. Each event has an indication as to whether the user has requested it interactively or whether it is performed by the system automatically as a side-effect, e.g. to adapt image and text to each other. We are currently investigating the generation of descriptive figure captions, which describe the image generated verbally (see HARTMANN *et al.* in [4] for first results). These descriptions include the current viewing direction and the usage of graphical techniques. Figure captions – as can be found for example in textbooks – are based on the information maintained by the *Context Expert* and comment especially those changes which have been performed by the system automatically.

Layout Layer

The Layout Layer (corresponding to the *Layout-Manager* in Figure 3), is not a central instance, managing all layout problems. Instead instances of each class (recall Figure 4) manage their own layout and distribute the space for subordinate instances.

With this policy class *Illustrator* knows how many *3D Models* and *Networks* are present and assigns space to them. This assignment, includes a rectangular area for usage and a tolerance area which can be used if necessary without informing the superior instance. Only if an instance cannot cope at all with the assigned space, the superior instance (see the *has-a-*

relations in Figure 4) is asked to redistribute space. This hierarchical mechanism turns out to be effective (in terms of speed) and flexible.

An example may clarify this strategy. If the user interacts intensively with textual information at one side, requesting explanations, the network is allowed to “grow“ a bit and the nodes involved may even grow a little bit more (recall Figure 3 with the different width of nodes). This can even result in small overlaps between text and graphics display. This is tolerable because overlaps are seldom and the exact extent of the graphics is – due to its irregular shape – difficult to consider. If some limit in the growth of a net is reached and information can only be presented at the expense of other information, the user recently interacted with, the *TextArea*-instance is informed. To circumvent the problem, it can move a node to another network or extend the network. Such a change, however, is not incremental and should not occur often, because it is expensive for the system and irritating for the user.

5 IMPLEMENTATION

The implementation was carried out on medium range Silicon Graphics Workstations. This platform allows to experiment with models of a “realistic“ complexity. The system uses OPEN INVENTOR™, an object-oriented graphics library is aimed at interactive applications. For our purpose it is crucial that an extensible class library is provided with classes for event-handling and the interactive manipulation of objects.

6 CONCLUDING REMARKS

We described the ZOOM ILLUSTRATOR which combines interaction facilities with automated techniques to support high-level-interaction. Automatic support is necessary for the annotation of objects, the basic layout and coordination between images and text.

The architecture of the ZOOM ILLUSTRATOR has been presented. Whenever possible, it is generic so that at least the higher level classes should be reusable for similar purposes. The architecture has been described in terms of the SRM although only a subset of the terms and components have a counterpart in the system. The interaction with textual information could be easily described. OO-Design is well-suited to state “who“ is responsible for which behavior and to define relations among classes. Furthermore, it naturally leads to an OO-Implementation. The OO-Design usually results in a vertical structure with hierarchical relations. However, OO-Design tends to bring up lots of classes the overall structure of which is easily lost, especially when few *has-a* or *inherits* relations exist between them.

The horizontal layer structure of the SRM has advantages in terms of its ability to explain this overall structure. This becomes obvious, for instance, in the description of the layout policy. The distribution of the responsibility to different classes is useful for the implementation and maintenance of the software. The overall strategy, however, can be better described within the Layout Layer of the SRM. While the experts presented in the SRM lend themselves to be designed as classes in a OO-system, the layer structure is orthogonal to OO-Design. The tasks being supported by different layers are distributed to many classes in an OO-Design. In fact, each part of an illustration performs its own layout calculation and presentation. However, even the layer structure can be exploited to enhance an OO-Design, because it allows to structure the methods of the classes as to the layers for which they perform a service (many classes have *presentation*-methods, *layout*-methods,...).

To summarize this discussion: The SRM as well as the OO-Design have their advantages and can be used in a complementary way. None of these notations, however, can replace the other in its entirety. The SRM is superior in its ability to describe the system’s architecture at a higher level of abstraction, whereas the OO-notion is superior to describe at a lower level how an OO-system works. In unison with one another they lead to a structured system description. It remains an open question whether or not the generic architecture of the SRM can (or should) be described in an OO-way similar to the OO-architecture included in this paper.

Future Work

The architecture presented encompasses text and graphics. Text presented can be clearly assigned to the parts of the graphics it refers to via lines. The disadvantage, however, is that both images and text must be processed by our visual system. The ZOOM ILLUSTRATOR would benefit from the incorporation of speech output, which is especially suited for verbal explanations which are not as closely connected to the graphics. This, however, leads to new challenging coordination problems.

The SRM gives us new insights in the system and reveals room for improvement. The inclusion of a *User Expert*, for example, would enable us to tailor the presentation to a specific user. Such a personalization is extremely helpful, if the educational aspect of the system is extended (the systems asks questions and answers them) and the system adapts its behaviour to the user's reactions. The current system does not use intelligent reasoning and generation. Thus the flexibility of the generation is limited. The ZOOM ILLUSTRATOR would profit from the use of natural language generation techniques to tailor the presentation of textual information.

ACKNOWLEDGEMENT

My thanks go to Thomas Strothotte who has encouraged this work and has initiated my interest in Fisheye Techniques. The author wishes to thank Alf Ritter, who implemented the initial version of the ZOOM ILLUSTRATOR. Furthermore my thanks go to Knut Hartmann, Ian Pitt as well as to the reviewers for commenting on the paper.

REFERENCES

[1] G. Booch (1994)
Object-Oriented Analysis and Design with Applications, The Benjamin Cunnings Publishing Company (Redwood, Ca, 1994), Second Edition

[2] J. Dill, L. Bartram, A. Ho, and F. Henigmann (1994)
"A Continuously Variable Zoom for Navigating Large Hierarchical Networks", *Proc. of IEEE Conference on Systems, Man and Cybernetics*, November, 386-390

[3] G. W. Furnas (1986)
"Generalized Fisheye Views", *Proc. of ACM SIGCHI'86* Conference (Boston, Ma, April), 16-23

[4] K. Hartmann, B. Preim, and T. Sommerfeld (1997)
„Bildunterschriften zur Erklärung räumlicher Zusammenhänge", in: *Proc. of Elektronische Sprachverarbeitung* (Cottbus, Germany, August), to appear

[5] B. Preim, A. Ritter, and T. Strothotte (1996)
"Illustrating Complex Phenomena: A Semi-Interactive Approach", in: *Proc. of Visualization in Biomedical Computing*, Lecture Notes in Computer Science, Vol. 1131 (Springer-Verlag, Berlin), 23-32

[6] B. Preim, A. Ritter, and T. Strothotte (1997)
„Coherent Zooming of Illustrations with 3D Graphics and Text", in: *Proc. of Graphics Interface* (Kelowna, Canada, May), 105-113

[7] M. Rüger, B. Preim, and A. Ritter (1996)
"Zoom Navigation: Exploring Large Information and Application Spaces", *Proc. of Advanced Visual Interfaces* (Gubbio, Italy, May), 40-48

[8] W. Wahlster, E. André, W. Finkler, H-J. Profitlich, and T. Rist (1993)
"Plan-Based Integration of Natural Language and Graphics Generation", in: *AI-Journal* 63: 387-427