Chapter 6
Case Studies of the Model-Based Approach

6.1 Introduction

At present, we have used OPOSSUM with the Relational, MOOSE, and E-R data models, in each case using a variety of visual models and metaphors. The visual model of graphs has proved to be meaningful for all three data models. We have also used OPOSSUM beyond schema visualization, as the core of query and object visualization tools for ZOO. This section describes how OPOSSUM can be used for many kinds of schemas and other data. Special attention is given to how OPOSSUM can be used to support abstraction.

OPOSSUM is currently used by scientists in Biochemistry and Soil Sciences to design schemas for their databases or experiments. The feedback in all cases has been very positive, especially with respect to OPOSSUM's ease of use: people are able to learn the system and then organize and layout large schemas with hundreds of classes within a few hours. This informal feedback from our users has proved very valuable; as our user base expands we intend to do more organized usability assessments to better tailor the operation of OPOSSUM to its intended users.

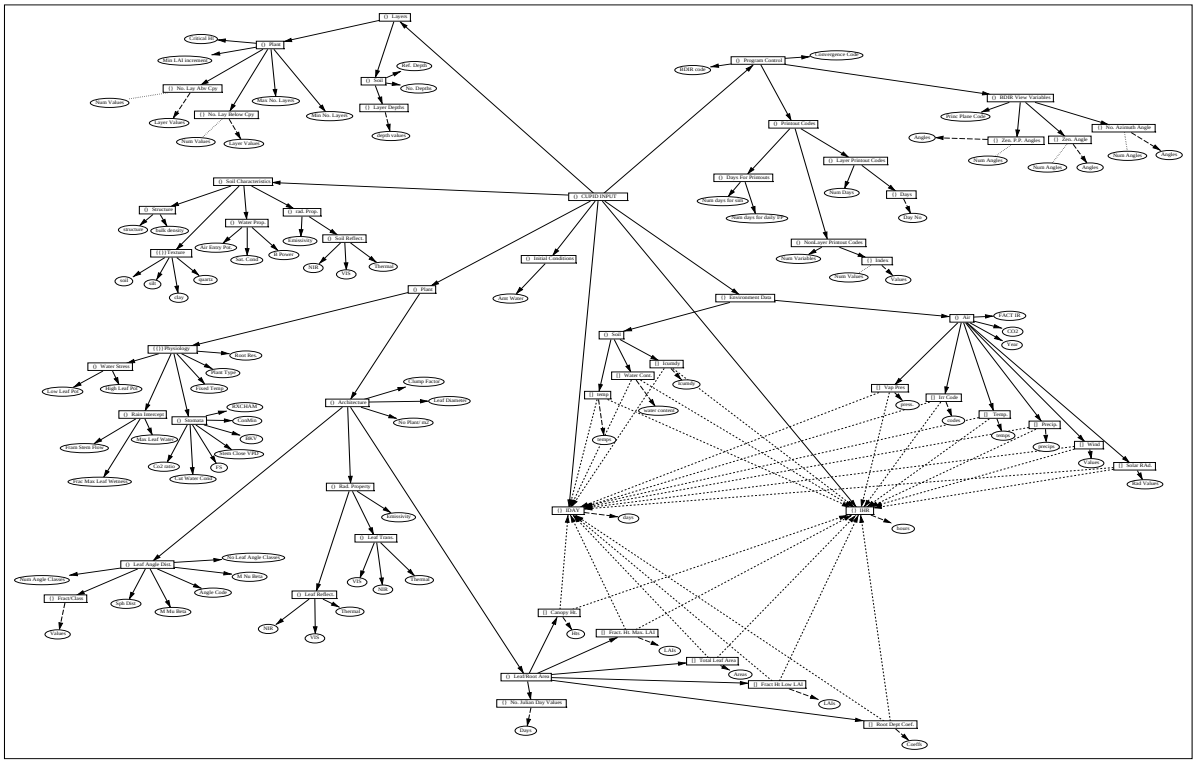

Figure 6.1: The input of the Cupid simulation model as a MOOSE schema, created by OPOSSUM.

6.2 Cupid, MOOSE, and a Graph Visual Model

A group of soil scientists has used OPOSSUM to layout a MOOSE schema describing the parameters of Cupid, a FORTRAN simulation model of plant growth [IL92, ILH92]. The input part of the Cupid schema alone has 159 classes (see Figure 6.1). The visual model is that of a directed graph, where Nodes and Edges represent classes and relationships, respectively, Node shape and a text field represent class kind, and the Edge's line pattern indicates the type of the relationship. The tool has brought substantial improvements in the scientists' work. Before OPOSSUM, the soil scientists' only reference to Cupid was the input data file to the Fortran program, which had grown increasingly fractured and confusing over the years. They now use the visual schema of Figure 6.1 as their reference in thinking about the model, planning experiments, and explaining experiments to other scientists.

6.3 A Relational Schema as a Graph

Another group of scientists, working on NMR research, have used OPOSSUM to develop a very large Relational schema (over 250 tables and more than 2000 attributes) for describing their experiments. They have used another graph-based visual metaphor, with rectangular relation Nodes, oval attribute Nodes, and vertical placement representing relation-attribute relationships. In addition, primary keys are captured by a darker color and foreign keys relationships are represented as Edges. An example of a small piece of this schema is shown in Figure 6.2. The NMR researchers have found OPOSSUM very useful in helping them design their large schema since it enables them to see its overall structure and easily navigate around it.

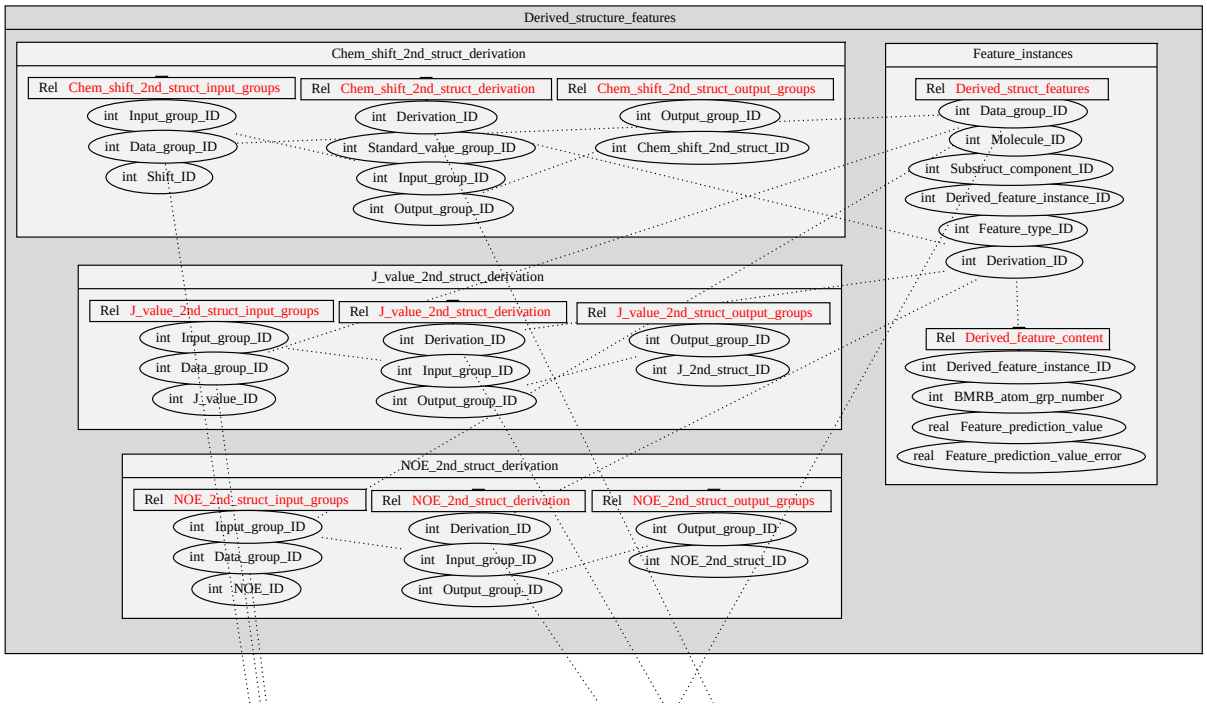

Figure 6.2. Part of the NMR Relational schema.

```
Root Resistance − P
    Has−Part From Physiology

Roughness − P
    Has−Part From Structure

Sand − P
    Has−Part From Texture

Sat . Cond. − P
    Has−Part From Water Prop.

Soil − ()
    Has−Part To integer As No. Depths
    Has−Part To real As Ref. Depth
    Has−Part To Layer Depths
    Has−Part From Layers

Soil − ()
    Has−Part To Icumdy
    Has−Part To Water Cont.
    Has−Part To temp
    Has−Part From Environment Data

Soil Characteristics − ()
    Has−Part To Texture
    Has−Part To rad. Prop.
    Has−Part To Structure
```

Figure 6.3. Part of the Cupid schema, as a textual list.

6.4 Other Visual Models and Other Data

While the two previous examples used visual models based on directed graphs, Opossum is capable of supporting other types of models. One example is containment for expressing parent-child relationships, something used in Grouping (see Section 6.6). Another example is textual lists, whose linear nature allows them to be sorted. Figure 6.3 shows a portion of the Cupid schema visualized using a textual list visual model, sorted by class name. This visual model and metaphor is basically a 2-level outline, representing MOOSE classes as MainPoints, and relationships as SubPoints. MainPoints are pairs of labels with a fixed X location and free Y location. SubPoints include labels beneath and to the right of both the source and the destination of the Relationship. Constraints ensure placement of the SubPoints below the MainPoints that they connect.

The variety of visual models for schemas also prove useful for queries and data. Directed graphs work well for queries; queries can be considered as pieces of schema with various qualifications and operators attached. To visualize them, one only needs to extend a visual model for schemas to include representations for the qualifications and operators. In a similar manner, schema visualizations can be used as a template for representing instances of data. Such a template works well for browsing, but is less useful in showing many instances at once (unless one has a very good automatic layout program). To show many instances at the same time, one can use a textual list, or design a visual model especially for the data. For example, the visual model and metaphor shown in Figure 6.4 captures x-y charts by mapping attributes of the data to position in the plane, and shape of chart ticks.

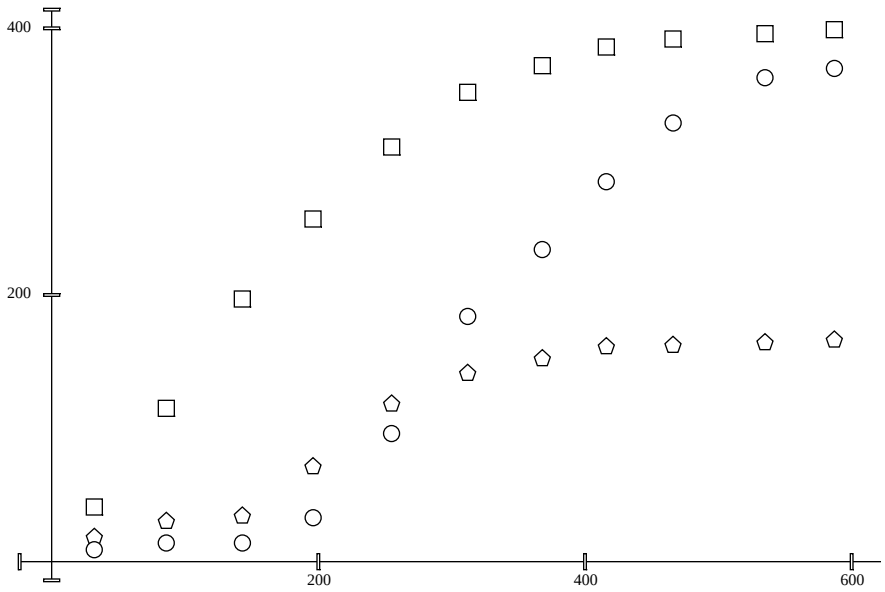

Figure 6.4. A sample visual model and metaphor for representing data as an X-Y chart.

The data model has one concept, the tuple, with two integer attributes, and a third attribute with a range of four values. The visual model and metaphor map tuples to a visual concept called a point; the integer values of the tuple are mapped to the X and Y locations of the point, and the other attribute is mapped to one of four shapes. The axes and ticks on the axes only exist in the visual model, and thus represent aesthetic information. Figure 6.4 demonstrates how OPOSSUM can be used to visualize information that is very different from the directed-graph-like structures seen in the other examples in this thesis.

6.5 A Case Study of Model Creation: The Entity-Relationship Model

As a test of the flexibility of OPOSSUM, the system was used to create models and a metaphor for working with Entity-Relationship diagrams. The instigator was a professor in the Computer Sciences department at the University of Wisconsin-Madison who wished to have a tool for students in introductory database courses. Previously this professor had hired a student to create a custom E-R tool using tcl/tk, but that tool never reached a stage where it could be used due to problems capturing the behavior and appearance of E-R diagrams. Using OPOSSUM, however, a working tool was created within 10 hours (including several hours fixing bugs brought out by the particular needs of the E-R model). After receiving feedback from the professor, two more hours were spent to remove a few remaining deficiencies.

An example schema created using the E-R model in OPOSSUM is shown in Figure 6.5. The visual model is fairly standard, with rectangular entities, oval attributes, and diamond relationships. Notable features of this model are aggregates and is-a relationships. Aggregates allow a relationship and all the entities associated with it to be considered a single entity that may be a part of other relationships. Aggregates are indicated by a box with dashed borders surrounding the relationship, its participating entities, and all of their attributes. In Figure 6.5, the “Works-On-Project” relationship is aggregated.

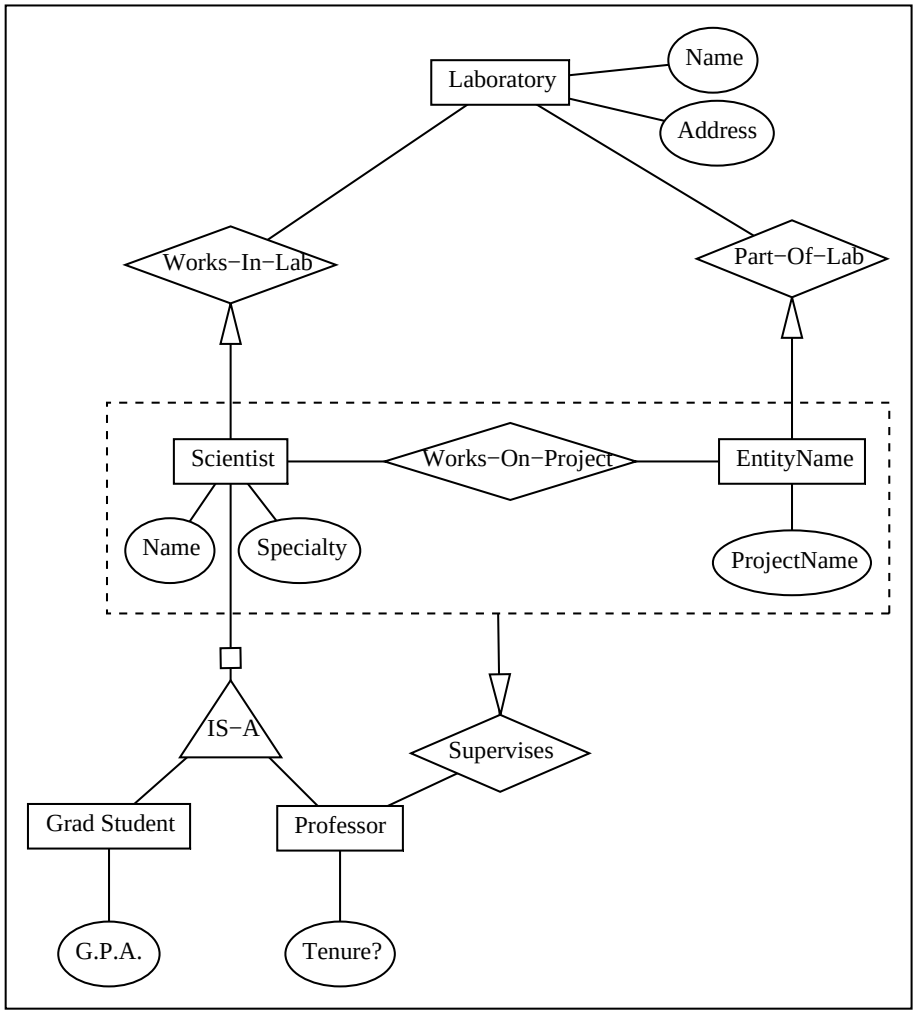

Figure 6.5. A sample E-R schema created using OPOSSUM

Is-A relationships are used for describing situations where different entities have common properties. In the visual model, they are represented as triangles with a marked edge leading to the parent entity, and unmarked edges leading to each child. Figure 6.5 shows that both Grad Students and Professors can act as Scientists working on a project, but the two entities have different attributes, and only a Professor can supervise a project and all the scientists who work on it.

This implementation of the E-R model using OPOSSUM highlights both the advantages and disadvantages of the system. On the positive side, it was possible in a very short time to create a tool supporting complex behavior (something that had been difficult to do with a custom tool). On the negative side, it highlighted three important limitations of our approach:

- The underlying formalism has a single-valued nature: concepts have a fixed number of attributes, and attributes may only have single values. Because of this, in order to support n-ary relationships, it was necessary to divide the relationship into two different concepts: the relationship itself, and the connection between a relationship and an entity. On the screen, the relationship diamonds are separate instances from the arcs connecting them to entities, and they must be created separately. This makes working with the model less natural that it might otherwise be.

- The current implementation of constraints does not support quantification. The professor requesting this tool wanted a different appearance for arcs and relationship boxes between weak entities and their owning entities. Quantification was needed because the relationships and arcs should base their appearance on whether the owner attribute of the entity at one end of the relationship was the same as an entity at the other end. Lacking quantification, constraints could not be written to ensure this. As a result, users must change the appearance manually in order to have a correct schema.
- When editing attribute values, the user is presented with a menu listing the internal names of all the attributes. These internal names are originally specified in the meta-creator model description, and while they are suitable for their role there, they are not the most informative for the average user. It would be nicer if there were additional names for attributes for use in the Edit menu (and other places where the user is shown the name of an attribute).

None of these problems prevented the tool from being used, or from satisfying the needs for an E-R diagram editing tool, but they do make this tool more cumbersome to use. There is nothing about the underlying formalism that prevents changes to fix these problems, however, so in the future OPOSSUM should overcome these limitations.

6.6 A Case Study of Layout: Implementing Grouping with OPOSSUM

In this section, we focus on how OPOSSUM supports abstraction using the layout techniques described in section 4.7, illustrating in the process several features of the system. The ease with which this is done provides a case study of the flexibility and extensibility of our tool.

6.6.1 Problem

During schema exploration, a user often wants to see both the high-level structure of schemas and their fine details. The Grouping approach addresses this problem through visual abstraction: the addition of a visual construct called a *Group* which clusters subgraphs and allows the subgraph to either be seen in its entirety, or represented as a single visual item. As an example, Figure 6.6 presents the same schema as in Figure 6.1, only partitioned into rectangular groups. Figure 6.7 shows the schema at its highest level of abstraction, with only 11 groups and nodes. Finally, Figures 6.8 and 6.9 show the internal structure of one and two top-level groups, respectively, allowing study of their details.

During a session of schema exploration, a user may focus on different parts of the schema and view them at several levels of abstraction. This should be achievable by simply collapsing or expanding groups, e.g., moving from Figure 6.7 to 6.8 and then to 6.9 should involve expanding one group and then collapsing it and expanding two others, respectively. Moreover, each group expansion or collapse should require very little user interaction, e.g., a single keystroke or mouse click.

Such ‘automatic’ switching back and forth between two visualizations can cause layout problems. When a group collapses, e.g., from Figure 6.8 to 6.7, the entire graph should move closer together so that there is better utilization of the freed-up space. Likewise, when a group expands, the rest of the graph should move away to make space for the group's expanded representation; otherwise, there may be node/group overlaps with undesirable semantics or aesthetics. Managing such layout side effects caused by changes in the group footprints is a key challenge for any system that implements Grouping.

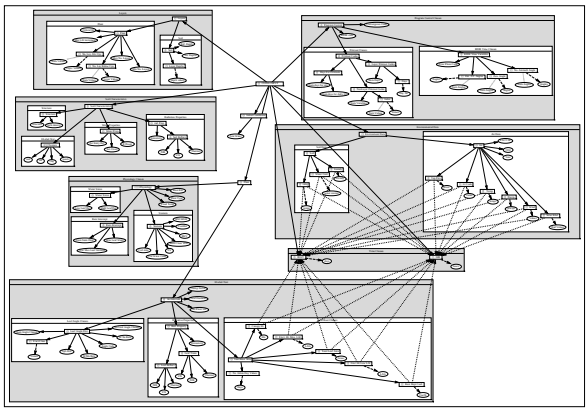

Figure 6.6. The CUPID input schema, Grouped.

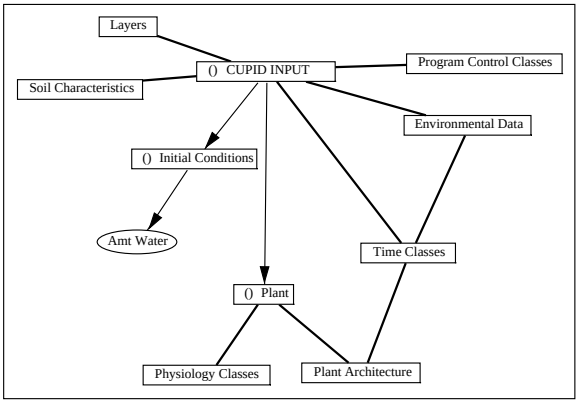

Figure 6.7. The same, abstracted.

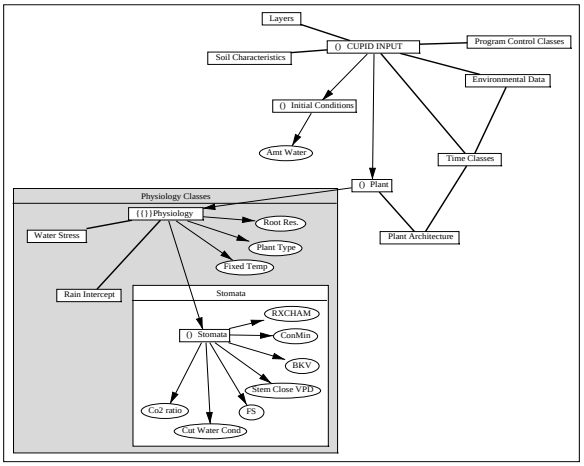

Figure 6.8. The same, partially abstracted.

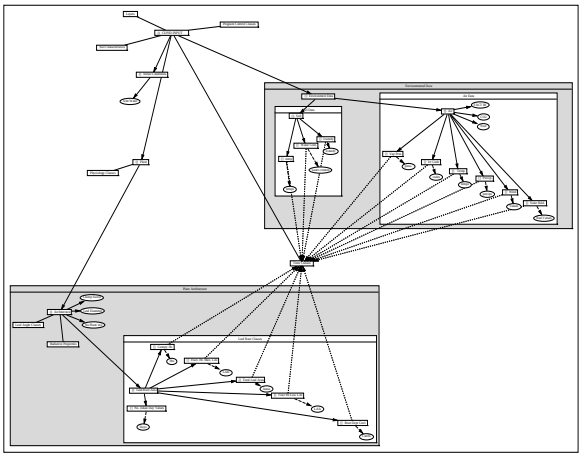

Figure 6.9. The same, differently abstracted.

6.6.2 Grouping as an Example of Mixed Visual Metaphors

Grouping has already been seen in several systems, e.g., QBD* [ACS90], SUPER [AAD+94], GUIDE [WK82] (in a limited form) and others [TWB+89]. Below we describe how it is realized using OPOSSUM, with little effort, as an example of a mixed metaphor.

We take the typical graph visual model (Chapter 4) and introduce a new concept called a *Group*. In addition to all their necessary display attributes, both Node and Group are given a ParentGroup attribute, which takes values among the instances of Group in any schema. Also, a Group has an expanded and a collapsed visual representation, the choice between which is controlled by the value of a Representation attribute, which makes the resulting metaphor mixed. In the expanded representation, a Group appears as a large box with a label at the top. In the collapsed representation, a Group appears the same size as a Node.

In addition to the above, the model has several constraints to ensure the visual integrity of a graph with Groups, among which the most important are the following (expressed in English for readability):

- In the expanded representation of a Group, if the footprint of a Node or Group *c* is within the footprint of a Group *p* then *c* is a member of *p*.
- In the expanded representation of a Group, all its members are visible.

- In the collapsed representation of a Group, all its members are invisible.
- The footprint of a Group is no smaller than the footprint of all its members.
- When a Group's expanded location changes, the locations of its members change by the same amount.

Using the layout methodology described in Section 4.7, expanded Groups, collapsed Groups, and Nodes are prohibited from partially overlapping each other (though containment is permitted). The system is instructed to maintain two safe layouts, one for expanded Groups and Nodes, and another for collapsed Groups.

Given the multiple representations of groups and the potential invisibility of group members, visibility of edges becomes an issue as well. For this, the visual model is enhanced with a GroupEdge concept. GroupEdges connect collapsed groups when those Groups have members that are connected, as for example, in Figure 6.7. Constraints similar to the above are defined to handle the visibility of GroupEdges and regular Edges.

6.6.2 A 'Demo' of OPOSSUM with Grouping

In this subsection, we demonstrate how the above visual model achieves the desired Grouping functionality, and we describe how Grouping appears to the OPOSSUM user. The system operates as described in Chapter 5; the schema manager reads in the appropriate model and metaphor descriptions, and the user can then manipulate schema graphs equipped with Groups, as for example, in Figure 6.6.

Given the enhanced visual model, OPOSSUM's column of buttons will include a buttons for Group and GroupEdge. By selecting the first of those buttons, a subsequent click starts the process for the generation of a Group, which is the specification of a rectangle area on the screen. Again, this behavior is dictated solely by information that exists in the visual model, i.e., the fact that the concept Group has a region with unspecified boundaries.

Upon the creation of a Group as above, by the immediate enforcement of constraint (1) in the previous subsection, the group members are automatically determined to be those covered by the specified area, as expected. The new group's ParentGroup, if any, is also determined at that time. Any subsequent changes to the membership of the group are achieved by simply moving nodes in and out of the group's boundary, again as a result of enforcing constraint (1).

During schema creation and modification, a user may also need at some point to move a group and expect for its members to follow along. Similarly, a user may modify the size of a member of a group (e.g., as a result of an expansion) and expect that the boundary of the parent group will also change, if necessary, to continue to enclose the member. The correct behavior is observed as a result of enforcing constraints (4) and (5).

Finally, as discussed above, the most critical feature of Grouping is the ability to collapse and expand Groups. This is accomplished by changing the value of the Representation attribute of the concept Group. To switch from one representation to the other, a user has simply to click on the group while the 'Edit' button is selected, choose Representation from the appearing menu, and then modify its value to that of the desired representation. The constraints of the model (including (2) and (3) above) will ensure that the desired representation appears on the screen without any further action from the user. Moreover, all the layout problems that may occur with the representation change are handled by a heuristic described in Section 4.7.

From all the above, it should become clear that OPOSSUM can be customized to provide complex functionality like Grouping with no specialized effort, simply through the appropriate definitions of models and metaphors. Not only is the desired appearance and behavior of Groups obtained, but most tasks involved with using Groups can be accomplished with very few actions on the part of the user. We believe that Grouping is a

testament to the flexibility and extensibility of our approach to visualization.

6.6 Summary

The variety of the above examples demonstrate that OPOSSUM has the flexibility to be used as a visualization and editing tool for schemas and many other kinds of information. Furthermore, this flexibility has proven very helpful to the users of this system in managing large schemas.