Chapter 7
Related Work

7.1 Introduction

As mentioned in earlier chapters, the main contribution of OPOSSUM is its comprehensive support for (a) declarative definitions of data and visual models and metaphors in a way that is explicit, extensible, and external[9] to the database system, (b) multiple and mixed metaphors for visualization, and (c) enhancements of schema visualizations with personal information and aesthetic preferences. To the best of our knowledge, support for (a) alone makes OPOSSUM the only implemented database schema manager whose appearance and behavior can be customized by the user. OPOSSUM is also unique in supporting (c), personal information.

Some of the ideas in (a) and (b) can be found in other systems as well. In this section, we present an overview of the spectrum of related (implemented or paper) systems and pinpoint their differences from OPOSSUM. We divide systems into four categories:

- schema visualization and manipulation tools (schema managers);
- query or data visualization tools that use schemas as templates;
- query or data visualization tools unrelated to schemas;
- user interface tools, which generate separate stand-alone visualization tools.

For the last two categories, the notion of metaphor is generalized to capture a mapping from a visual model to arbitrary internal domains, not simply data models. Table 7.1 summarizes the overall comparison that follows, indicating for each system surveyed whether or not it supports any of the features in (a) and (b). In the few cases of unnamed systems, the primary reference to them is used as an identifier. When several systems are presented as a group, the characterization of the group in parenthesis is used as an identifier. The abbreviations VM and DM are used to indicate “visual model” and “data model” respectively.

7.2 Schema Visualization and Manipulation Tools

Most database systems have a fixed, hard-coded visual model and metaphor. Although textual representations are most common, more advanced visual metaphors have also been used extensively. These may be classified as follows:

- Tables, which use rows and columns to indicate database structure, as with QBE [Zlo75] and other systems [BH86, HR85, KM89, dO94, OMO89, Weg89].
- Diagrammatic presentations, such as E-R-like Diagrams [ACS90, CS89, EL85, LSN89, LCC94, Miu91, SCT91, WK82, ZM83] and other directed and non-directed graphs [BFG94, BH86, Cre89, GWJ91, HGPN92, KM84, LCT+90, PK94, PdB92, SM94, YDS+87].
- Icons (pictures) that represent a concept or action [CCL91, HGR94, KH86, THTI90].

Although diagrammatic representations seem to be the most popular, the persistence of other approaches further demonstrates the importance of flexible schema visualization.

[9] We view all three ‘ex’ properties as equivalent.

Systems	Model Definition		Metaphor Definition		Metaphor Choice	
	Declarative	Extensible	Declarative	Extensible	Multiple	Mixed
Schema Tools						
OPOSSUM	x	x	x	x	x	x
(Most systems)						
(Flexible Schema Vis. Tools)					x	
[Eng92]				x	x	
Schema Template Tools						
DOODLE	x	x	x	x	x	
(Report Generators), (Forms Systems)			x	x	x	
(OODBMS Visual Tools)		VM Only		x	x	
Query/Data Tools						
Hy+ [Eig93]						x
Q-PIT [Ben94], PRIMA [Sch92]					x	
[Cat94c]					x	x
Tool Generators						
UIDE [Suk93]	x	x	partly	x		
HUMANOID [Sze93]	VM Only	VM Only				
Binnacle [Bed89]		VM Only		x		x
R1 [Hou89]				x		x
(Most OODBMS Tool Generators)		VM Only		x		x
[Coo90]	DM Only	DM Only				
[Car92]						

Table 7.1. Comparison of OPOSSUM with other Related Systems.

There are a few systems that provide multiple metaphors for visualizing schemas. ISIS-V [DZ86], Sidereus [AACO88], and SUPER [AAD+92] can all present schemas (or queries) as either directed graphs or series of tabular dialog boxes. IDDS [Nor92] supports schema design by people with different levels of expertise through textual or graphical visual models (as well as an interactive dialog asking users yes/no questions about classes they create). WINONA [RK94] provides two unusual visual models for schemas (a circular wall and a three dimensional hierarchy) and discusses several others. While these systems provide a certain amount of choice, the alternatives are predefined and hard-coded.

Finally, [EC92] suggests visualizing schemas using a data visualization tool that permits user-specified mappings of information to icons in a 2-D space. This system's flexibility, however, stops at metaphor definition; the data and visual models are hard-coded and even the resulting schema visualization cannot be manipulated by the user.

7.3 Schema Template Tools

The idea of using schema visualizations as templates for query specification or data presentation goes all the way back to QBE [Zlo75] and GUIDE [WK82] and has been routinely incorporated in many systems and prototypes. Many of these systems use the same visualization mechanisms for the schema proper and have been mentioned in Section 7.2. In this subsection, we discuss tools that use the schema only as a template.

Although not yet implemented, conceptually the most advanced of these systems (and, in fact, the one that comes closest to OPOSSUM) is DOODLE [Cru93]. It offers a visual declarative language that allows users to define with pictures OODBMS queries or visualizations of database objects. DOODLE supports the concept of *visual language*, which is a declarative definition of either a visual concept or a relationship between visual concepts, along with what that concept or relationship represents. Thus, it defines the visual model and metaphor together, and the former is not separable from the latter. Similarities between OPOSSUM and DOODLE include the declarative specification of models and metaphors, the ability to visually specify models and metaphors, and the ability to have multiple visual languages, i.e., multiple metaphors. The disadvantages of DOODLE over OPOSSUM include its inability to support mixed metaphors (it only allows one visualization per class per visual language), and more importantly, the fact that DOODLE has not yet been implemented. The advantages of DOODLE include its ability to specify visual constraints and data visualizations by example and the fact that the expressive power of its language is well studied.

On a simpler level, the relational report generators and form tools found in most commercial database systems (and some research prototypes [KN87, Row85]) allow users to specify metaphors for schema visualizations to be used for query and data presentation. These metaphors are usually defined either interactively or with a declarative language, and many of them may co-exist (i.e., one may use multiple forms or report definitions for the same data). The data and visual models in such systems are mainly textual, however, and are not extensible.

Finally, there is a multitude of schema template visualization tools in Object-Oriented database systems, which use class methods to procedurally define object visualizations (ADAM [PaQK92], DEED [Rad92], ODDS [FM92], OdeView [AGS90], and [SCSM92]). They all present the user with a single, mixed metaphor, where each object can be seen in one of many ways. A limitation of these systems is that there is no mechanism to deal with co-existing contradictory or ambiguous visualizations.

7.4 Query/Data Visualization Tools

There are a few systems that offer choice in metaphors for query and/or data visualization, though they do not support flexibility in the definition of the used models and metaphors. For data visualization, Hy+ [CM93, Eig93] deals with graph-type data and provides a mixed metaphor to accommodate a grouping abstraction similar to that of Chapter 5. The main difference is that grouping in OPOSSUM is simply a special case of the general methodology and mechanism of the system, while in Hy+ it is a hard-wired feature. Two other such systems are Q-PIT [BM94], which proposes allowing user-defined mappings of tuples and attributes to visual concepts and attributes in a 3-D space (although there is no concrete suggestion on how this mapping would be accomplished), and PRIMA [Sch92], which allows query results to be viewed at four levels of abstraction. For query visualization, [CCS94] describes a powerful framework that uses both mixed and multiple metaphors; queries may be specified in a diagrammatic, form, icon, or hybrid fashion, with procedural transformations between them.

7.5 Tool Generators

In our discussion of tool generators (user interface tools), we view them as a means to define models and metaphors for the tools that they generate, and focus on the properties of their definitional power. We distinguish two categories of tool generators, those unrelated and those related to database systems. Most general user interface tools specify visual models procedurally, though a few, such as HUMANOID [SLN93] and UIDE [SFG93], allow description of visual models through expressions in formal modeling languages. HUMANOID does not allow any description of a data domain, and all data is encapsulated in the visual concepts. UIDE permits declarative definitions of the data and visual domains, though its 'metaphor' is defined as correspondences between operations on visual model objects (such as a click of the mouse) and actions on data model object (such as incrementing a value).

There are several user interface tools designed to support a specific database system and hence a single data model. Binnacle [BM89] and R1 [HP89] are two similar systems for the nested relational data model, which allow users to procedurally specify the metaphor and behavior of a database interface through an extensible description of a finite state automaton. They are different in that Binnacle allows extensible descriptions of the visual model as well, whereas R1 does not. There are also several user interface tools for object-oriented data models (DUET [OZL94], FaceKit [KN89], O2Look/ToonMaker [BMP+92], Picasso [RKS+90]), most of which like stand-alone OODBMS visualization tools, allow the procedural definition of a single, mixed metaphor, and associated visual model. Two notable exceptions are [Coo90], which is able to deal with several object-oriented and semantic data models by capturing their common characteristics, and [CSF92], which deals with a specific object-oriented model and uses a single, hard-coded visual model and metaphor to automatically generate form and menu-based interfaces from the database schema.

Chapter 8
Conclusions

8.1 Summary of Results

This thesis has studied the area of visual management of database schemas and other similar information. Visual schema management in needed due to the increasingly novice user base of schemas, growing complexity of schemas to be managed, and the proliferation of graphical user interfaces that use schemas as templates for many operations. The history of database interfaces shows a trend towards visual representations of increasingly complicated database information, and surveys indicate that sophisticated visualization will be more important in the future.

The best way to address these needs for visualization is through a flexible approach not tied to any particular data model or visual style. This is because:

- There is no "best" way to visually represent any given data structure. Each visual style has advantages and disadvantages, each highlights some aspects of the underlying information more than others. For a given narrow context, one visual style may be better than others, but these contexts will change frequently.
- Each user understands the schema in a different way. As a result, each user may want to see the schema displayed differently, and furthermore they may want to add details of their understanding to the visual representation.

Hence, a visual schema manager must be very flexible, in how the schema is represented (as a whole, or in parts), in the visual styles used, and in permitting each user must be allowed to add their own information to the schema to help them better understand what they see.

The core of this thesis is the formal framework to support visualization flexibility. The framework brings visualizations to the same level as schemas by describing both in the same manner, and providing the following capabilities:

- Declarative specification of models and metaphors, so that schema visualization is realized externally and extensibly instead of through hard-wired code.
- Separation of the visual and conceptual (data model) realms, so that information in the latter can easily be represented using many different styles in the former.
- The ability to combine visual models and metaphors, so that different parts of a schema can be represented using different visual styles.
- The ability to use excess information capacity in the visual model to capture aesthetic and personal information for each different user.

These capabilities are demonstrated in the OPOSSUM schema visualization tool, illustrating the feasibility and value of the formalism in a real system. OPOSSUM works with many data and visual models, permitting users to manage large and complex schemas.

8.2 Future Work

There are several avenues of future work down which interesting problems remain. These may be divided into issues involving the formalism, issues of the OPOSSUM tool, and general questions about the approach as a whole.

With respect to the visualization formalism, one interesting area is extending the formalism to other domains outside of static visual information. The idea of mapping a data model to the realm of sound, animation, smell, or some other domain seems reasonable. The key to this is finding appropriate groups of constructs in the domain from which to built the concepts of the "sensorization" (a visualization in some other sensory domain). An important part of the sound or animation domains is the notion of time: both are streams of data changing over time. Time will need to be accounted for in any exploration of these domains. Time is also an interesting issue for other data types. The formalism described here considers only static data. It would be useful to extend it to manage time-varying data as well, whereby metaphors could be defined that map not only attribute values, but also integrals and derivatives of those values.

One limitation with the formalism in its current form is that while it describes appearance of visual concepts well, their behavior is less easily specified. Certain aspects of behavior are regulated by the constraints, but the current implementation of the constraint language is too limited to adequately control behavior.

Another area of further work on the formalism is study of the testability of metaphors. A metaphor may be checked for certain aspects of correctness, completeness, and quality. It would be useful to find out to what degree such testing is feasible and, if so, whether it proves useful to the model and metaphor designer.

Finally, the set of visual constructs used in the model could be extended. The current set, text-display, region, line, and bit-map, have proved useful, but occasionally limited. Other valuable constructs might be curves, splines, and multi-line pieces of text. It would also be interesting to extend the visual constructs to capture three dimensions, e.g. having a polyhedron region.

With respect to OPOSSUM, four areas require further work. The first is that a more formal evaluation of the system's usability is needed. This could be used to determine any problems with OPOSSUM's generic approach, and to compare various visual styles to show the advantages and problems of each. The second area is meta-creation. Allowing model and metaphor definition using a meta-model in OPOSSUM is feasible, but the resulting meta-schemas are very large and complex. Improved meta-models should be sought, and other possibilities considered. One alternate approach for specifying visual models is the "demonstrational" method, whereby the user draws a template on the screen from which the system derives the important characteristics of a visual concept. It would be useful to see if OPOSSUM could be extended to support demonstrational model definition. The third area of OPOSSUM requiring more work is its extension to work with queries and data. Prototypes of query specification tools and data browsers have been built using OPOSSUM, but it has yet to be determined if OPOSSUM and the formalism have sufficient power to capture these tasks. Finally, the implementation of the constraint language needs further enhancement. Right now, constraints cannot express existential quantification, and they can only capture universal quantification in a limited manner.

A final area for future work is a more general question about the generic approach to visualization: how good can a generic visual editor be? Is there some point where hard-wired knowledge of the domain is needed? Or can a generic visualization tool provide all the functionality of a more specialized tool? The answers to these questions will determine the range of usefulness for generic visualization tools.