Journal of Intelligent Information Systems, 3, 263–298 (1994)

Foundations of Visual Metaphors for Schema Display

EBEN M. HABER haber@cs.wisc.edu
YANNIS E. IOANNIDIS yannis@cs.wisc.edu
MIRON LIVNY miron@cs.wisc.edu
Department of Computer Sciences, University of Wisconsin, Madison, WI 53706

Abstract. Many aspects of database systems have been improved by Graphical User Interfaces (GUIs). One area that has not received adequate attention in GUI research is the visual presentation of schemas, despite the increasingly important role that schemas play in database design and operation. Schema visualizations are valuable for viewing and manipulating both the schema and the information captured by it. In this paper, we describe a framework that formalizes the process of visualizing database schemas or any similar structured information. It is based on the concepts of a *data model* capturing schemas, a *visual model* capturing visualizations, and a *visual metaphor* that defines a mapping between the two models. This formal description of the mapping between a schema and its visualization permits straightforward declaration of visual metaphors, and provides criteria to evaluate metaphors as to their ability to correctly visualize a schema. Given a visual metaphor, the framework divides visual information into that which has meaning relative to the data model, that which has meaning to the user but not to the data model, and that which is only aesthetic. This separation permits better use of aesthetic information, resulting in richer visualizations. As a whole, we believe that the formalism provides the foundations on which better schema visualization tools can be built.

Keywords: Database system, schema, user interface, visualization of structured information, metaphor

1. Introduction

Graphical User Interfaces (GUIs) are playing increasingly important roles in Database Management Systems (DBMSs): improving ease of use through more intuitive operation and increasing information bandwidth from computer to user through visual display of information. Research in this area has touched upon many aspects of DBMSs, including visual DBA tools (Benjamin and Lew, 1986), non-textual specification of queries (Batini, et al., 1991; Cruz, 1992), displaying and querying multimedia and visual information (Egenhofer and Frank, 1988; Gupta, Weymouth, and Jain, 1991; Leong, Sam, and Narasimhalu, 1989; Yoon, et al., 1987), creating visual representations of complex data (Flynn and Maier, 1992; King and Novak, 1992; Maier, Nordquist, and Grossman, 1986; Mamou and Medeiros, 1991), browsing through databases (Agrawal, Gehani, and Srinivasan, 1990; Motro, 1986; Stonebraker and Kalash, 1982; Tsuda, et al., 1990), and form-based data entry (King and Novak, 1987).

One area that has not received adequate attention is schema visualization. Many DBMS GUIs allow non-textual specification and browsing of schemas, but they focus on other aspects of DBMS operation and not schema visualization per se. The majority of these systems are limited to a single visual metaphor; they do not permit visualization flex-

ibility. Issues of efficient, flexible, and usable schema display are important, yet have received little attention in existing systems.

A database schema describes the conceptual structure of data stored in a DBMS. Traditionally, schemas have been visualized textually as expressions in a data definition language and have been used for database design. Schema visualization is crucial in the database design process, where an unintuitive display can result in misunderstandings and design errors. In some systems, schemas play a role beyond that in database design. Many DBMS GUIs also use the schema as a template for operations such as querying, browsing, data entry and display (e.g., GOOD (Paredaens and Van den Bussche, 1992), GORDAS (Elmasri and Larson, 1985), GUIDE (Wong and Kou, 1982), QBE (Zloof, 1975), QDB* (Angelaccio, Catarci, and Santucci, 1990), VILD (Leong, Sam, and Narasimhalu, 1989), and many others). Our own work focuses on scientific databases and experiment management systems (in the context of the ZOO Experiment Management System (Ioannidis and Livny, 1992; Ioannidis, et al., 1993). In these systems, the schema describes the structure of experimental data, and as such can capture the interrelationships between different parts of an experiment. A good visualization of a schema can aid users in better understanding the experiment and its significance, useful not only for human-computer interactions but also for collaborations among scientists. We are working to improve visualization of schemas to assist their use in these various roles.

To better understand and support the process of schema visualization, we develop a formalism with the following main elements: 1) a *data model* that captures schemas, 2) a *visual model* that captures visualizations, and 3) a mapping between data and visual models, referred to as a *visual metaphor*. In the interface community, the term *metaphor* has been used in a variety of ways, generally describing a transformation between abstract and visual information (Batini, et al., 1991).[1] The abstract information of concern to us is the database schema. Examples of visual metaphors for schemas include directed graphs, E-R diagrams, and textual tables. Clearly, these metaphors have different characteristics, and would be useful in different circumstances. In general, there is no ideal metaphor, thus metaphor choice is important. Unfortunately, no general metaphor selection criteria exist. For the specific case of database schemas (and similar structured information), our formalism is intended to provide a framework for flexible use, definition, and evaluation of visual metaphors. This formalism permits the declarative definition of models and metaphors, provides criteria for the identification of incorrect metaphors, and presents some guidelines for metaphor comparison. It supports mixed metaphors, the use of different visual metaphors for different parts of a single schema, establishing when and how metaphors may be mixed. Finally, the formalism allows the richness of some visual models to be used to capture information that is meaningful to the user but is beyond the database schema. We have begun implementation of a schema visualization tool based on this formalism. The tool displays and allows users to manipulate schemas and subschemas using different metaphors, and in the future will allow users to define models and metaphors dynamically. While our work is oriented towards database schemas, the formalism is also applicable to the visualization of any structured data that conforms to our definition of data models (which is found in the next section).

The rest of this paper is organized as follows. Section 2 describes data and visual models. Section 3 defines visual metaphors. Section 4 discusses correctness and quality of visual metaphors. Section 5 discusses mixing of metaphors. Section 6 describes how the metaphor framework may be used to capture information that is conceptually meaningful or aesthetically pleasing to the user, but is unrelated to the database schema. Section 7 describes related work in the area. Section 8 outlines future work and offers conclusions.

2. Data and Visual Models

For completeness, before presenting the definition of data and visual models, we briefly review some basic definitions and notations of binary relations which will be used later in the definition of visual metaphors. A binary relation r from set A (its *domain*) to set B (its *range*) is denoted $r : A \rightarrow B$. The relation r is called a *function* if for each $a \in A$ there is at most one $b \in B$ where $(a, b) \in r$; it is called *total* if for each $a \in A$ there is at least one $b \in B$ where $(a, b) \in r$; it is called *injective* if its inverse is a function; and it is called *onto* if its inverse is total. Injective functions are sometimes called 1-1 functions. Binary relations can be unioned, in which case, their domains, their ranges, and the sets of pairs in the relations are unioned, respectively. For a function r, we use $r(a) = b$ instead of $(a, b) \in r$. Functions may be applied to sets of values; if $A' \subseteq A$, then $r(A')$ is equal to $\{r(a) \mid a \in A'\}$. We also use r^{-1} to denote the inverse of a function r, which may return a subset of A if r is not 1-1. Finally, for a function r, the notation $r(a)$ is valid even if there is no $b \in B$ where $r(a) = b$. In that case, when applying a set-valued function on $r(a)$, the empty set is returned.

2.1. *Problem Formulation*

By definition, a *schema* describes the conceptual structure of some information in a database, specified using the primitives of some *data model* (for clarity, such a schema will henceforth be referred to as a *data schema*). We are interested in the process of creating a visualization of a data schema. For this, we introduce the notion of a *visual model*. Like a data model, it describes the structure of some information, though its primitives are visual. As with data schemas, any visual representation that conforms to a visual model will be called a *visual schema*. Visual schemas are used for two basic operations: presenting data schema information in a visual form, and allowing creation or manipulation of a data schema through changes to a visual schema.

Using the above notions, the problem of visual representation of data schemas may be stated formally as follows. Given a data model $\mathcal{D}$, let $S(\mathcal{D})$ denote the set of valid data schemas that can be constructed based on that model. Similarly, let $S(\mathcal{G})$ denote the set of visual schemas that can be constructed based on a visual model $\mathcal{G}$. The sets $S(\mathcal{D})$ and $S(\mathcal{G})$ are defined to be the *information capacities* (Miller, Ioannidis, and Ramakrishnan, 1993) of the data model $\mathcal{D}$ and the visual model $\mathcal{G}$, respectively. In order to create visual

schemas of data schemas, we require a binary relation between $S(\mathcal{D})$ and $S(\mathcal{G})$, whose specific properties depend on the intended use of the visual schemas. Specifically,

1. If a visual schema of $\mathcal{G}$ is used only to view *any* data schema of $\mathcal{D}$ in its *entirety*, then an onto function must exist of the form $f : S(\mathcal{G}) \rightarrow S(\mathcal{D})$, so that every data schema can be represented visually.

2. If, in addition, a visual schema of $\mathcal{G}$ is also used to update a data schema of $\mathcal{D}$, then a total onto function must exist of the form $f : S(\mathcal{G}) \rightarrow\!\!\!\!\!\rightarrow S(\mathcal{D})$, so that every visual schema can be uniquely interpreted as a data schema.

Clearly, not all such functions f that satisfy the above properties are useful. Many are arbitrary mappings, with no obvious correspondence between the data schema and the visual schema. Our goal is to establish a relationship between the members of $S(\mathcal{D})$ and $S(\mathcal{G})$ so that when users view a visual schema, they can infer the data schema to which it maps. Thus, f should be derived from a correspondence between the features of the data and visual models, which would enforce a structural similarity between data schemas and visual schemas. This correspondence is a *visual metaphor* and is formally introduced in Section 3. A formal description of the features of data and visual models is given in the next subsection.

We should mention at this point that the above problem formulation has been motivated by the very similar problem of mapping schemas and data between different data models in a heterogeneous database system (Miller, Ioannidis, and Ramakrishnan, 1993). The specific similarities and differences between the two problems are beyond the scope of this paper.

2.2. *A Formalism for Data/Visual Models and Schemas*

In this subsection, we present a meta-model that can capture a large and interesting class of data and visual models. Using this meta-model, we can describe the features of any model in this class and discuss the relative information capacity of any pair of models. Example models described in this meta-model are given in the following subsection.

DEFINITION 1 *Every data or visual model $\mathcal{M}$ can be seen as a sextuple $\mathcal{M} =< \mathcal{P}, \mathcal{A}, \mathcal{V}, \mathcal{Q}, \mathcal{R}, \mathcal{C} >$ defined as follows:*

- $\mathcal{P}$ *is a finite set of identifiers for the types of primitives in $\mathcal{M}$. Each such type P is associated with a (possibly infinite) set of globally unique ids $\mathcal{I}(P)$ that can be used to identify primitives of type P.*

- $\mathcal{A}$ *is a finite set of identifiers for property names of primitive types in $\mathcal{M}$. Each element of $\mathcal{A}$ is of the form $P.A$, where $P \in \mathcal{P}$ and A captures some attribute that all primitives of type P should have.*

- $\mathcal{V}$ *is a (possibly infinite) set of identifiers for values of properties of primitive types in $\mathcal{M}$. Each element of $\mathcal{V}$ is of the form $P.A == v$, where $P.A \in \mathcal{A}$ and v captures*

some value that the $P.A$ attribute may have. For certain attributes, the values v may be drawn from the sets $\mathcal{I}()$ of primitive ids.

$\mathcal{Q}$ *is a function* $\mathcal{Q} : \mathcal{P} \rightarrow 2^{\mathcal{A}}$, *indicating for each primitive type* $P \in \mathcal{P}$ *the set of attribute identifiers in* $\mathcal{A}$ *that correspond to* P.

$\mathcal{R}$ *is a function* $\mathcal{R} : \mathcal{A} \rightarrow 2^{\mathcal{V}}$, *indicating for each attribute* $P.A \in \mathcal{A}$ *the set of value identifiers in* $\mathcal{V}$ *that can be assigned to it. To capture the set of actual values instead of their identifiers (e.g.,* v *instead of* $P.A == v$*) the function* $\mathcal{R}^*$ *is used, where* $\mathcal{R}^*(P.A) = \{v \mid P.A == v \in \mathcal{R}(P.A)\}$

$\mathcal{C}$ *is a finite set of constraints, i.e., rules that must be satisfied by any schema expressed in* $\mathcal{M}$. *These constraints are formulas in some prespecified language* $\mathcal{L}$ *and use elements that refer to identifiers in* $\mathcal{P}$, $\mathcal{A}$, $\mathcal{V}$.

Note that, by the way $\mathcal{A}$ and $\mathcal{V}$ were defined, if $P \neq P'$ then $\mathcal{Q}(P)$ and $\mathcal{Q}(P')$ are disjoint, and if $P.A \neq P'.A'$ then $\mathcal{R}(P.A)$ and $\mathcal{R}(P'.A')$ are disjoint. Also note that primitives are essentially complex objects, since some of their attributes can take values that are primitives themselves.

Most of the common data models fall naturally in this meta-model. For example, consider the relational model. It has *relations* and *attributes* as its primitive types, each *relation* has a *name*, and each *attribute* has a *name*, a *type*, and a *relation* with which it is associated. Fully specified examples of data and visual models may be found in Section 2.4. The meta-model may be enhanced with several additional characteristics of models in a straightforward way, e.g., with an identifier for the name of each instance of the model, but we avoid that for simplicity of presentation.

As defined above, a data schema or visual schema may be considered as an instantiation of a data or visual model, respectively. This is formally defined as follows:

DEFINITION 2 *A schema S of a model $\mathcal{M}$ is defined as follows:*

- *For every* $P \in \mathcal{P}$, *there is a finite set* $[P] \subseteq \mathcal{I}(P)$ *of primitives of type* P *that appear in schema* S.
- *For every* $P.A \in \mathcal{A}$, *there is a total function* $[P.A] : [P] \rightarrow [\mathcal{R}^*(P.A)]$, *which determines the value of the* $P.A$ *attribute for every primitive in* $[P]$. *The* $[\mathcal{R}^*(P.A)]$ *set is defined such that, if* $\mathcal{R}^*(P.A) = \mathcal{I}(P')$, *for some* $P' \in \mathcal{P}$, *then* $[\mathcal{R}^*(P.A)] = [P']$. *Otherwise,* $[\mathcal{R}^*(P.A)] = \mathcal{R}^*(P.A)$.
- *For every* $c \in \mathcal{C}$, *there is a constraint* $[c]$, *constructed from* c *by replacing every* $P \in \mathcal{P}$ *by* $[P]$, *every* $P.A \in \mathcal{A}$ *by* $[P.A]$, *and every* $P.A == v$ *by* v. *All these constraints are satisfied by the schema.*

In the following table, we summarize the notations introduced in Definitions 1 and 2:

Notation	Explanation
$\mathcal{P}$	The set of primitive types
P	A primitive type
$\mathcal{I}(P)$	The set of identifiers for primitives of type P
$[P]$	The set of primitives of type P in a schema
$\mathcal{A}$	The set of attributes for all primitive types in $\mathcal{P}$
$P.A$	The A attribute of primitives of type P
$[P.A](p)$	The value of the $P.A$ attribute of the primitive p
$\mathcal{V}$	The set of values for all attributes in $\mathcal{A}$
$P.A == v$	The element v as a value of the attribute $P.A$
$\mathcal{Q}(P)$	The set of attribute identifiers of primitives of type P
$\mathcal{R}(P.A)$	The set of value identifiers of the $P.A$ attribute
$\mathcal{R}^*(P.A)$	The set of values of the $P.A$ attribute
$\mathcal{C}$	The set of constraints of a model
$[c]$	The instantiation of a constraint $c \in \mathcal{C}$ for the primitives in $[P]$ and attribute values $[P.A]$ in a schema

2.3. Creating Visual Models

Creation of data models is a classical database problem that is beyond the scope of this paper (Batini, Ceri, and Navathe, 1992). In this subsection, we concern ourselves with creating suitable visual models. There is an important difference between the two kinds of models. Data models capture abstract organization of information. Their primitive types, attributes, and values are determined by the information the model captures. Visual models, however, must reflect not only the information to be organized, but also the medium in which the models are expressed. Specifically, visual model primitive types reflect both the information to be shown and the medium, while the possible attributes and values of a primitive type are determined by the medium alone. For example, consider a visual model used to display directed graphs. Any such model would likely have primitive types corresponding to *nodes* and *edges*. If the model were oriented toward a monochrome ASCII terminal, these primitive types and their attributes and values would be very different from a similar model intended for a color bitmapped display system. The ability to use colors, shapes, lines, and patterns would vary widely between the two. In general, the number and semantics of visual model primitive types are determined by the information that must be displayed, but the precise composition of the primitive types is determined by the medium.

Because data models are used to represent abstract information, their types of primitives may be chosen arbitrarily based on some conceptualization of the world. On the other hand, visual model primitives must be visualizable. Therefore, visual primitive types must be constructed using only certain visual building blocks. Motivated by our

involvement in developing a scientific Experiment Management System, we are concerned with visual models to be displayed on color bitmapped workstations as these are commonly available to scientists. To build visual models for this medium, we have chosen the basic visual constructs described in the following table. The choice of these constructs is somewhat arbitrary. They are not formally defined in this paper, but the interested reader may find a formal discussion of visual constructs elsewhere (Foley, et al., 1990). In the table below, location is a complex attribute consisting of the spatial coordinates of the system. For the region, text-display, and picture-display constructs, it is assumed to be the location of their center.

Construct	Attributes
region	shape, location, orientation, size, background-color, background-pattern, boundary-width, boundary-color, boundary-pattern
line	source-location,dest-location, width, color, pattern
text-display	text, font, location, orientation, size, and color
picture-display	picture, location, orientation, size, and color

Visual primitive types are defined as *compositions* of the above primitive types or other, previously defined, visual primitive types. The attributes of a visual primitive are the attributes of all of its components, possibly renamed to avoid any naming conflicts. Since compositions often have a large number of attributes, in our examples we have omitted many visual attributes to make the examples more manageable.

To make the appearance of a composition coherent, it will usually be necessary to include constraints relating the attributes of its different components. For example, if a box with a piece of text in the center were required, a composition of a *region* and a *text-display* would be defined as a primitive type, and a constraint would require the value of the location attribute of *text-display* to be the same as the value of the location attribute of *region*. These constraints regulate the appearance of visual primitives, and are called *composition constraints* to distinguish them from other, more semantically focused constraints.

2.4. *Example Data and Visual Models*

Consider a very simple semantic data model, supporting *entity-classes* that may be mutually related with binary *relationships*. Each entity-class has a *name* and a *kind*. The two possible kinds are 'simple' (such as the class of integers or the class of character strings) or 'compound' (user-defined classes). Each relationship has a *name*, a *card-ratio* of '1:1', '1:N', 'M:1', or 'M:N', and two entity-classes with which it is associated. This data model is the sextuple $\mathcal{D} = < \mathcal{P}_\mathcal{D}, \mathcal{A}_\mathcal{D}, \mathcal{V}_\mathcal{D}, \mathcal{Q}_\mathcal{D}, \mathcal{R}_\mathcal{D}, \mathcal{C}_\mathcal{D} >$, where $\mathcal{C}_\mathcal{D} = \emptyset$, and the primitive types in $\mathcal{P}_\mathcal{D}$, their attributes in $\mathcal{A}_\mathcal{D}$, and their corresponding value sets as determined by $\mathcal{R}^*_\mathcal{D}$ are given in the following table.[2]

Primitive Type (P)	Attribute ($P.A$)	Attribute Values ($\mathcal{R}^*(P.A)$)
entity-class	name	text
	kind	{simple, compound}
relationship	name	text
	card-ratio	{1:1, 1:N, M:1, M:N}
	from-class	$\mathcal{I}$(entity-class)
	to-class	$\mathcal{I}$(entity-class)

Note that the set of values of an attribute has several possibilities: an infinite predefined set (e.g., text), an enumerated set (e.g., {1:1, ..., M:N}), or the set of all instances of a primitive type (e.g., $\mathcal{I}$(entity-class)).

Similarly, consider a very simple visual model that supports directed graphs. We define the primitive types to be *nodes* and *edges*, the former a combination of a *region* and a *text-display*, and the latter a combination of a *line*, a *text-display*, and two *nodes*. This visual model is the sextuple $\mathcal{G} = < \mathcal{P}_\mathcal{G}, \mathcal{V}_\mathcal{G}, \mathcal{A}_\mathcal{G}, \mathcal{Q}_\mathcal{G}, \mathcal{R}_\mathcal{G}, \mathcal{C}_\mathcal{G} >$, where the primitive types in $\mathcal{P}_\mathcal{G}$, their attributes in $\mathcal{A}_\mathcal{G}$, and their corresponding value sets as determined by $\mathcal{R}^*_\mathcal{G}$ are given in the following table. For simplicity, only a subset of the attributes is shown.

Primitive Type (P)	Attribute ($P.A$)	Attribute Values ($\mathcal{R}^*(P.A)$)
node	shape	{square,oval}
	location	plane-points
	size	{100 pixels}
	color	{blue, red}
	label-text	text
	label-color	{black}
edge	source-location	plane-points
	dest-location	plane-points
	color	{black, blue, yellow, green, orange}
	from-node	$\mathcal{I}$(node)
	to-node	$\mathcal{I}$(node)
	label-text	text
	label-color	{black}

There are four constraints in set $\mathcal{C}_\mathcal{G}$ that all schemas of $\mathcal{G}$ must satisfy. Two of them are composition constraints, related to the relative positioning of visual constructs within each primitive type. The remaining two are somewhat more interesting, determining the location of *edge* primitives in terms of the *nodes* they connect. Using simple Horn-clauses, we show these constraints, which indicate that the location of the source (resp. destination) of an edge is the same as the location of the from-node (resp. to-node) of the edge:

$\forall e \in$ edge, source-location(e) = location(from-node(e)), and
$\forall e \in$ edge, dest-location(e) = location(to-node(e)).

3. Visual Metaphors

3.1. *Definitions and Notation*

A visual metaphor is defined as a correspondence between some of the features of a data and a visual model, i.e., elements in $\mathcal{P}, \mathcal{A}, \mathcal{V}$. A metaphor induces a mapping between data schemas (instances of the data model) and visual schemas (instances of the visual model). Basing the schema mapping on the feature correspondence helps produce visual schemas that, when viewed, allow the user to deduce the underlying data schema (Section 2.1). Consider a data model $\mathcal{D} = < \mathcal{P}_\mathcal{D}, \mathcal{A}_\mathcal{D}, \mathcal{V}_\mathcal{D}, \mathcal{Q}_\mathcal{D}, \mathcal{R}_\mathcal{D}, \mathcal{C}_\mathcal{D} >$ and a visual model $\mathcal{G} = < \mathcal{P}_\mathcal{G}, \mathcal{A}_\mathcal{G}, \mathcal{V}_\mathcal{G}, \mathcal{Q}_\mathcal{G}, \mathcal{R}_\mathcal{G}, \mathcal{C}_\mathcal{G} >$. A metaphor will include correspondences between primitive types ($\mathcal{P}_\mathcal{D}$ and $\mathcal{P}_\mathcal{G}$), between attributes ($\mathcal{A}_\mathcal{D}$ and $\mathcal{A}_\mathcal{G}$), and between attribute values ($\mathcal{V}_\mathcal{D}$ and $\mathcal{V}_\mathcal{G}$). (The above define a correspondence between constraints as well, since constraints refer to elements of the $\mathcal{P}$, $\mathcal{A}$, and $\mathcal{V}$ sets.) These correspondences describe the meaning of visual model features with respect to the underlying data model. For example, given the data and visual models from Section 2.4, if a correspondence were defined between the primitive types *entity-class* and *node*, then every instance of a *node* in a visual schema would imply the existence of a *entity-class* in the data schema. To allow presentation flexibility, we permit correspondences to exist between multiple features in the visual model and a single feature in the data model (an example of this is given in the next section). This is possible only when the visual model has a greater information capacity than the data model (Section 2.1), which is almost always the case.

DEFINITION 3 *A metaphor T is an onto function from $\mathcal{G}$ to $\mathcal{D}$ (denoted by $T : \mathcal{G} \rightarrow \mathcal{D}$), which is the union of the following three onto functions:*

Function $T_p : \mathcal{P}_\mathcal{G} \rightarrow \mathcal{P}_\mathcal{D}$.
Function $T_a : \mathcal{A}_\mathcal{G} \rightarrow \mathcal{A}_\mathcal{D}$, *which is equal to* $\bigcup_{P \in \mathcal{P}_\mathcal{G}} T_a^P$, *where for each primitive type* P, $T_a^P : \mathcal{Q}_\mathcal{G}(P) \rightarrow \mathcal{Q}_\mathcal{D}(T_p(P))$ *is an onto function.* [3]
Function $T_v : \mathcal{V}_\mathcal{G} \rightarrow \mathcal{V}_\mathcal{D}$, *which is equal to* $\bigcup_{P.A \in \mathcal{A}_\mathcal{G}} T_v^{P.A}$, *where for each attribute* $P.A$, $T_v^{P.A} : \mathcal{R}_\mathcal{G}(P.A) \rightarrow \mathcal{R}_\mathcal{D}(T_a(P.A))$ *is an onto function.*

As mentioned above, all constraints in $\mathcal{C}_\mathcal{G}$ use elements that refer to identifiers in $\mathcal{P}_\mathcal{G}$, $\mathcal{A}_\mathcal{G}$, and $\mathcal{V}_\mathcal{G}$. We occasionally use the notation $T(c)$, $c \in \mathcal{C}_\mathcal{G}$, for the constraint constructed from c by replacing each element referring to an identifier x of $\mathcal{P}_\mathcal{G} \cup \mathcal{A}_\mathcal{G} \cup \mathcal{V}_\mathcal{G}$ by an element referring to the identifier $T(x)$. We also use the notation $T(\mathcal{I}_\mathcal{G}(P))$ to denote $\mathcal{I}_\mathcal{D}(T(P))$.

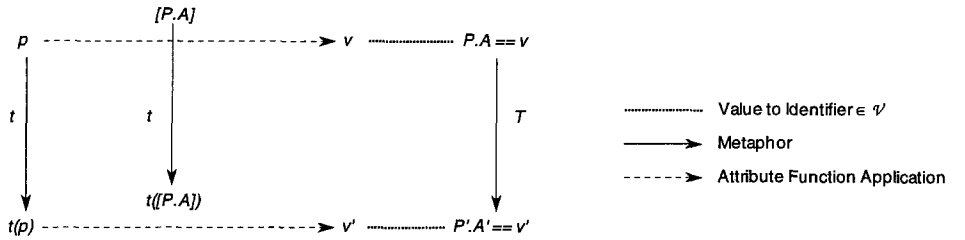

Figure 1. Commuting diagram between metaphors and attribute functions.

3.2. *The Induced Schema Mapping*

Given a metaphor as defined above, a mapping between data and visual schemas can be induced. Using this induced mapping, any data schema of $\mathcal{D}$ can be transformed to a visual schema of $\mathcal{G}$ in a manner that remains faithful to the metaphor.

DEFINITION 4 *Given a metaphor $T : \mathcal{G} \rightarrow \mathcal{D}$ and a visual schema of $\mathcal{G}$, T induces an onto function t from $(\bigcup_{P \in \mathcal{P}_\mathcal{G}} [P]) \cup \{[P.A] \mid P.A \in \mathcal{A}_\mathcal{G}\}$ onto the corresponding features of some data schema of D with the following characteristics:*

- $(\forall P \in \mathcal{P}_\mathcal{G})(\forall p \in [P])$ *if $T_p(P)$ is defined, then $t(p)$ is a primitive of type $T_p(P)$.*
- $(\forall P \in \mathcal{P}_\mathcal{G})(\forall P.A \in \mathcal{Q}(P))$ *if $T_a(P.A)$ is defined, then $t([P.A])$ is a function $[T_a(P.A)] : [T_p(P)] \rightarrow [\mathcal{R}^*_\mathcal{D}(T_a(P.A))]$ such that $(\forall p \in [P])$ the following holds:*

$$\begin{aligned} &\textit{if} \quad [P.A](p) = v \quad \textit{and} \quad T(P.A == v) = (P'.A' == v') \\ &\textit{then} \quad t([P.A])(t(p)) = v'. \end{aligned}$$

The first clause above states that primitives of the visual schema in $\mathcal{G}$ represent primitives in some data schema in $\mathcal{D}$ based on the type correspondence specified by T. The second clause states that, for every visual model attribute mapped by T_a, there exists a data model attribute whose values are determined based on the value correspondence T_v. Essentially, this is a commutativity requirement that is best shown in Figure 1. Based on the two clauses above, the induced function t determines a mapping from any visual schema in $\mathcal{G}$ to some data schema in $\mathcal{D}$. As with the metaphor T, the induced function t can be extended to include in its domain instantiations of constraints, $t([c])$ for $c \in \mathcal{C}_\mathcal{G}$.

3.3. *An Example*

To illustrate the above definitions, we present a metaphor T. This metaphor maps from a visual model similar to that in Section 2.4 to the data model discussed in the same section. The visual model has been supplemented with an additional primitive type, called a *blob*, which consists of a region and two text-displays, referred to as *label1* and

label2. Also, the *from-node* and *to-node* attributes of *relationship* have been extended to accept as values both *nodes* and *blobs*. This function T is defined in the table on the following page.

3.4. Discussion

A metaphor provides meaning to features of a visual model by establishing a correspondence between them and the features of a data model. The precise meaning is captured by the function T. For example, displaying a red oval *node* implies the existence of a compound *entity-class* in the data model. The use of the T function and its induced schema mapping t should produce visual schemas that users can correctly and unambiguously interpret. Depending on various properties of T, the visual schema may include features that do not carry any meaning and/or features that carry redundant meaning. Based on knowledge of T, users should be able to ignore the former and not be confused by the latter.

We would like to comment on the various properties of metaphors as they relate to the relative information capacity of a data and a visual model. As a minimum requirement, a metaphor has been defined as an onto function: if it were not onto, then some characteristics of a data model would not be captured visually; if it were not a function, then a single visual construct could have multiple meanings, and therefore could not be interpreted correctly.[4]

For a metaphor that is not total, some visual elements do not mean anything with respect to the data model. If a metaphor will be used only for retrieving a data schema, T does not have to be total. Under retrieval, only existing schemas will be visualized, so non-totality does not create any problem. Specifically, non-totality of T_p or T_v implies that some visual primitive types or some values of visual attributes respectively will never be used, while non-totality of T_a implies that the values of some attributes can be arbitrary. If a metaphor will be used for retrieving and updating a schema, T_p and T_a still do not have to be total. For T_a, the reasons are the same as above. A non-total T_p is permissible because visual primitive types that are not mapped by T_p may be used for presentation purposes and can be ignored when mapping the visual schema to a data schema. For each $P.A$ that is mapped by T_a, however, $T_v^{P.A}$ must be total. Otherwise, one could draw a visual schema that would not be translatable to a data schema.

To demonstrate the use of a non-total metaphor, consider a data model $\mathcal{D}$ capturing words in the English language as strings of letters from the Roman alphabet. A visual model $\mathcal{G}$ is constructed to visually present these words based on a straightforward metaphor that maps each display of a word to the word itself. Because the strings are visually expressed, $\mathcal{G}$ must also include information about typeface, size, color, letter spacing, and other visual characteristics of letters that carry no particular meaning, i.e., T_a is not total. Because of this, $\mathcal{G}$ has a greater information capacity than $\mathcal{D}$: the number of its visual schemas is equal to the number of English words (about 600,000) multiplied many times by the possible typefaces, sizes, and the other characteristics. Yet regardless of font or size, a visualization of a word carries an unambiguous meaning in the context of the metaphor.

x	$T(x)$
node	entity-class
node.label-text	entity-class.name
node.label-text==x	entity-class.name==x
node.color	entity-class.kind
node.shape	entity-class.kind
node.color=='blue'	entity-class.kind=='primitive'
node.shape=='square'	entity-class.kind=='primitive'
node.color=='red'	entity-class.kind=='compound'
node.shape=='oval'	entity-class.kind=='compound'
blob	entity-class
blob.label1-text	entity-class.name
blob.label1-text==x	entity-class.name==x
blob.label2-text	entity-class.kind
blob.label2-text=='P'	entity-class.kind=='primitive'
blob.label2-text=='C'	entity-class.kind=='compound'
edge	relationship
edge.label-text	relationship.name
edge.label-text==x	relationship.name==x
edge.from-node	relationship.from-class
edge.from-node==c	relationship.from-class==t(c)
edge.to-node	relationship.to-class
edge.to-node==c	relationship.to-class==t(c)
edge.color	relationship.card-ratio
edge.color=='green'	relationship.card-ratio=='1:1'
edge.color=='orange'	relationship.card-ratio=='1:1'
edge.color=='yellow'	relationship.card-ratio=='1:N'
edge.color=='blue'	relationship.card-ratio=='M:1'
edge.color=='black'	relationship.card-ratio=='M:N'

If a metaphor is not 1-1 then multiple visual elements have the same meaning with respect to the data model. For both retrieval and update, the implications of this are the same. If T_p or T_v are not 1-1 then there is a choice of visual constructs that can be used, which should be left to the user or resolved via some default mechanism. If T_a is not 1-1 then there is redundancy: multiple visual attributes capturing the same data attribute. By the nature of visual models, there is no issue of choice here: all attributes of a primitive must have some value in a visual schema, and therefore all those mapped to the same data attribute should be assigned *consistent* values based on T_v. This issue of consistency arises because of the redundancy semantics. For visual updates of schemas, if several visual attributes are mapped to the same attribute by T_a, as soon as the value of one of them is specified, the values of all others are uniquely determined.

Functions that are not 1-1 establish equivalence classes among the features of the visual model, i.e., several features have the same meaning. For example, in the metaphor of Section 3.3,

$$T_p(\text{node}) = T_p(\text{blob}) = \text{entity-class}$$

implies that a primitive of type *entity-class* may be represented equivalently as either a visual primitive of type *node* or one of type *blob*. Attribute correspondences are similar but more complex. A data model attribute may correspond to a set of visual attributes, some of which may come from the same primitive type. For example, in the same metaphor,

$$T_a(\text{node.color}) = T_a(\text{node.shape}) = T_a(\text{blob.label2-text}) = \text{entity-class.kind}$$

indicates the same choice of *node* or *blob* primitive types as above. In addition, it specifies redundancy: visual attributes *node.color* and *node.shape* are from the same primitive type, so they redundantly capture data attribute *entity-class.kind*. Value mappings are similar, but even more complicated, so they warrant two examples. First,

$$\begin{aligned} T_v(\text{edge.color} == \text{'green'}) &= T_v(\text{edge.color} == \text{'orange'}) \\ &= (\text{relationship.card-ratio} == \text{'1:1'}) \end{aligned}$$

demonstrates attribute value choice; two values have the same meaning with respect to the metaphor. Second,

$$\begin{aligned} T_v(\text{node.color} == \text{'blue'}) &= T_v(\text{node.shape} == \text{'square'}) \\ &= T_v(\text{blob.label2-text} == \text{'P'}) \\ &= (\text{entity-class.kind} == \text{'primitive'}) \end{aligned}$$

includes values from different primitive types (e.g. *node.color* versus *blob.label2-text*), paralleling the choice at the primitive type level, and different values for different attributes of the same primitive type (e.g. *node.color* and *node.shape*), indicating values for redundant attributes.

Figure 2 gives an example schema and a visual schema that could be produced by applying the induced mapping of the example metaphor in Section 3.3. (Due to limitations of the printing medium, color attributes cannot be displayed directly; instead they are indicated by the name of the color along the line of the edge.)

4. Judging Metaphors: The Good, the Bad, and the Ugly

Given a framework for creating visual metaphors, it is necessary to examine issues of metaphor correctness. We thus develop criteria that are useful in ensuring that a metaphor accurately presents information. There are issues beyond correctness, however, that affect how well metaphors visualize information. We discuss these issues of metaphor *quality* and their impact on visualization as well.

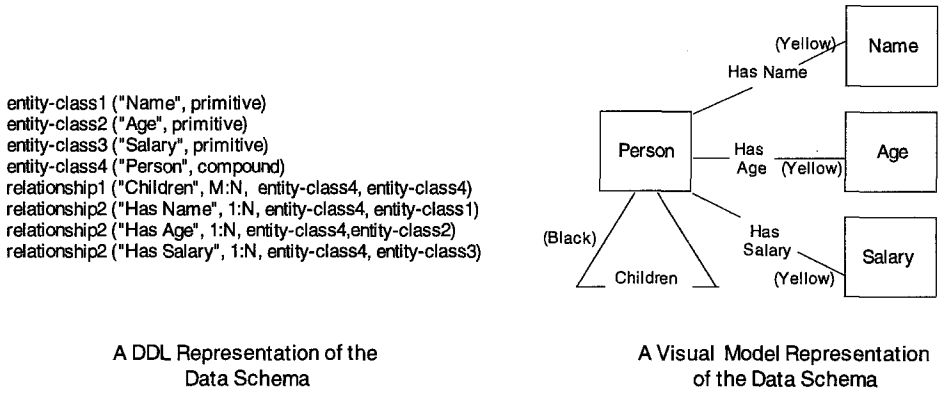

Figure 2. Example of a Metaphor Applied to a Schema.

4.1. Metaphor Correctness

We have already discussed the requirements for the relation T in order for it to be a valid metaphor based on the desired operational goals, retrieval and/or update. In addition, there are three other issues that affect the correctness of a metaphor. All three require some consistency between the metaphor and the visual model, the first in terms of allowed attribute values, and the second two in terms of constraints.

First, consider attributes whose values are primitives, e.g., the *from-class* attribute of *relationship* in Section 3.3. It is necessary that the type of such a primitive-valued visual attribute be consistent with the metaphor and with the type of the data model attribute to which it is mapped. Continuing the above example, the attribute *relationship.from-class* takes values of type *entity-class*. The primitive type *entity-class* is mapped to by more than one visual primitive type, specifically *node* and *blob*. As a result, it is necessary that the visual attribute *edge.from-node*, which maps to *relationship.from-class*, accept as values primitives of types *node* and *blob*. Specifically, if $\mathcal{R}^*_{\mathcal{D}}(P.A) = \mathcal{I}(P')$ for some $P' \in \mathcal{P}_{\mathcal{D}}$ then the following should hold:

$$\mathcal{R}_{\mathcal{G}}{}^*(T^{-1}(P.A)) \;=\; \cup_{P'' \in T^{-1}(P')} \mathcal{I}(P'').$$

For example, the metaphor of Section 3.3 satisfies the above since

$$\begin{aligned} \mathcal{R}^*_{\mathcal{G}}(T^{-1}(\textit{relationship.from-class})) \;&=\; \mathcal{I}(\textit{node}) \cup \mathcal{I}(\textit{blob}) \\ &=\; \cup_{P \in T^{-1}(\textit{class})} \mathcal{I}(P). \end{aligned}$$

Otherwise, it would not be possible to choose arbitrary *node* or *blob* representations for *entity-class* primitives.

Second, when there is redundancy in the metaphor, i.e., T_a is not 1-1 and different attributes of the same primitive type in $\mathcal{P}_{\mathcal{G}}$ are mapped to the same attribute of some

primitive type in $\mathcal{P}_\mathcal{D}$, the values of the former attributes must be consistent. This can be enforced with constraints in $\mathcal{C}_\mathcal{G}$. For example, in the metaphor presented in the previous section, the color and shape attributes of *node* are redundantly used to capture the kind attribute of entity-class. For the metaphor to be correct as defined in Section 3.3, the visual model must include the following constraints:

$$\forall n \in \text{node}, \quad \text{color}(n) = \text{'blue'} \Leftrightarrow \text{shape}(n) = \text{'square'},$$
$$\forall n \in \text{node}, \quad \text{color}(n) = \text{'red'} \Leftrightarrow \text{shape}(n) = \text{'oval'}.$$

Allowing any other combination of shape with color would permit visual schemas with no corresponding data schema. This would prevent the visual model from being used for updates of the data schema, since it would allow conflicting values of the *kind* attribute of the *entity-class*. For example, a visual schema with a blue oval *node* would have no meaning with respect to the metaphor.

Third, the constraints of the visual model should be such that no valid visual schema will map to an invalid data schema, and vice versa. This is ensured through a relationship between the constraints in the data model and those in the visual model. This relationship may be very complex, since one may perform inferences on a given set of constraints to derive additional constraints that are not explicitly specified. For the purposes of this paper, we take a simple approach and consider only some straightforward sufficient conditions for the consistency of constraints between the two models. Specifically, for two constraints c and c', c *subsumes* c' if the set of schemas that satisfy c is a subset of the set of schemas that satisfy c'. Consider the subset $\mathcal{C}_\mathcal{G}{}'$ of the constraints in $\mathcal{C}_\mathcal{G}$ that are mapped by the metaphor T. (Note that no composition constraint is among them.) If the visual model will be used for retrieval only, then the following should hold:

$$\forall c' \in \mathcal{C}_\mathcal{G}{}', \ \exists c \in \mathcal{C}_\mathcal{D}, \quad c \text{ subsumes } T(c').$$

This is sufficient to ensure that all data schemas have a corresponding visual schema. On the other hand, if the visual model will be used for updates as well, then the following should hold:

$$\mathcal{C}_\mathcal{D} \ = \ \{T(c) \mid c \in \mathcal{C}_\mathcal{G}{}'\}.$$

This is sufficient to additionally ensure that all visual schemas have a corresponding data schema. Note that the visual model may have additional constraints, those in $\mathcal{C}_\mathcal{G} - \mathcal{C}_\mathcal{G}{}'$, e.g., composition constraints or other constraints that are enforced for presentation purposes.

4.2. *Metaphor Quality*

A metaphor may be correct and none-the-less present information poorly. For example, it is conceivable to have a correct metaphor where T_a is not a function. Such a T_a would map the same visual attribute to more than one data attribute, so that each value of the former corresponds to a combination of values of the latter. We have decided,

however, that this introduces a level of visual complexity that is often uncomfortable and would result in confusing visual schemas in many cases. For example, consider the *relationship* primitive type of the data model in Section 2.4, enhanced with a *kind* attribute taking values 'part-of' and 'association'. Following the metaphor of Section 3.3, consider mapping the *edge.color* attribute to the combination of the *relationship.kind* and *relationship.card-ratio* attributes. Each of eight colors would map to a given combination of kind and ration, e.g., T('orange') = ('part-of', 'M:N'). Such a T_a relation is not strictly incorrect, i.e., a well defined mapping between visual and data schemas can still be derived with the desirable properties with respect to information capacity. We believe, however, that such a metaphor would be more difficult for most users to remember than one where two separate visual attributes are mapped to the two data attributes. We thus disallow non-functional T_a's.

Beyond functionality of T_a, other characteristics of *metaphor quality* also affect information presentation. We discuss three such traits that greatly affect metaphors: *information hiding*, which occurs when information is captured by the visual model but is not visible to the user, *visual ambiguity*, which happens when visual primitives of different primitive types or different values of the same attribute appear identical to the user, and *semantic ambiguity*, which exists when visual attribute values do not suggest the data attribute values they capture. The first two issues are concerned with the visual model alone, while the third regards the metaphor itself. There exist other issues of metaphor quality beyond those mentioned above, including intuitiveness, versatility, and emphasis. These are more difficult to quantify so they are beyond the scope of this paper.

In general, there are no universal rules about good user interfaces. Nevertheless, we believe there are certain desirable and undesirable characteristics of user interfaces in the context of our own work. We thus conclude the discussion of each of the metaphor quality traits above with comments on the trait's implications, and whether we believe these implications to be good or bad.

4.2.1. Hidden Information

Not all information captured by a visual model is visible to the user. This hidden information falls into two categories: transient and structural. Transient hidden information is not visible to the user but can become visible through some manipulation of the visual schema that leaves the underlying data schema unchanged. Consider the visual model described in Section 2.4 and the metaphor from Section 3.3. The *node* primitives have locations that may be anywhere on the plane, thus it is possible for two *nodes* to have the same location. The convention for such cases in a 2-D display system is to make one of the *nodes* invisible or partly visible, conceptually "behind" the other *node*. The *node* that is behind captures information, yet the user cannot see it. If the front *node* is moved, an operation that does not affect the underlying data schema, the back *node* becomes visible. Figure 3 demonstrates how one primitive can be partially or totally hidden by another.

This example of hidden information depends on the fact that the location attribute is *free*, not part of the metaphor. Freedom of other attributes can also result in transient

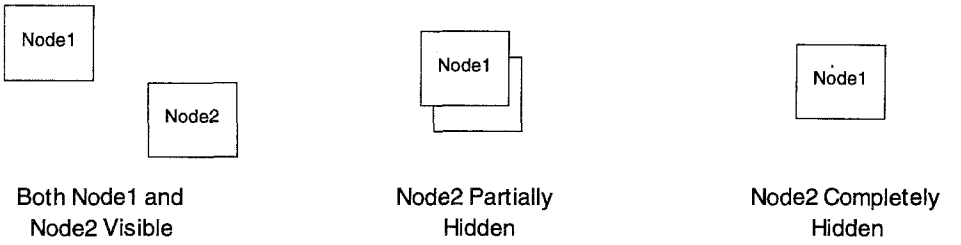

Figure 3. An example of transient hidden information.

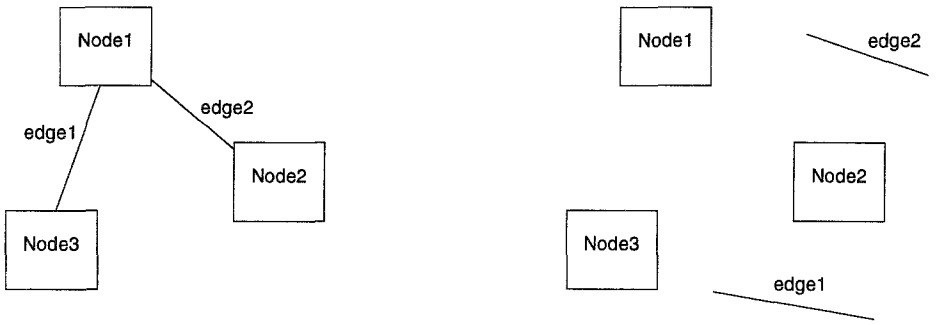

Figure 4. An example of structural hidden information.

hidden information. For example, if the size of a *node* were free, the *node* could be hidden by setting its size to zero.

Structural hidden information is information that a visual model captures but does not display. Consider the *edge* primitive type from the visual model in Section 3.1 and the metaphor in Section 3.3. It has two attributes, *from-node* and *to-node*, which are not directly shown. Their values are made visible by the constraints that define the location of the *edge*. Consider the result of removing this constraint from the visual model. The location of *edges* would no longer be constrained; an *edge* line could appear anywhere in the visual schema with no relation to the *nodes* specified by its *from-node* and *to-node* attributes. The visual model would still be valid; it would still capture the same information, but this information would not be visible to the user. Figure 4 demonstrates the appearance of a visual schema with and without the constraints which determine *edge* location. The same metaphor contains another example of structural hidden information. *Edges* are not directed, and as a result it is not possible to distinguish the *edge's from-node* from the *to-node*. Structural hidden information can occur whenever the appearance of one attribute is affected by another attribute or constraint. In the above example, the

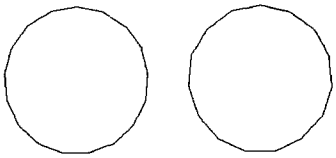

Figure 5. Polygons of 15 and 17 sides.

from-node attribute is made visible through a constraint that links it to another attribute, *source-location*.

We believe that structural hidden information should usually be avoided, whereas transient hidden information need not be. In many cases, a model that *cannot* visually display all the information it captures is undesirable. Temporarily invisible information, however, is not a problem as the hidden information can be made visible when necessary. In fact, transient hidden information can be a very useful tool for reducing clutter in a visual schema by hiding infrequently needed information.

4.2.2. *Visual Ambiguity*

Visual ambiguity occurs when a visual model contains two distinct primitive types (members of $\mathcal{P}_\mathcal{G}$) or two distinct values for the same attribute (members of $\mathcal{V}_\mathcal{G}$) that are visually indistinguishable. Two items are visually indistinguishable when a user viewing one cannot discern which of the two it is.

Objects with identical appearance are obviously visually indistinguishable. For example, it would be legal to define two different visual primitive types with the same attributes and values, and equivalent constraints. Metaphors could be correctly and unambiguously defined (since they depend on symbolic representations of the primitive types and their characteristics), but users might be unable to interpret schemas correctly, since instances of the two visual primitive types would appear the same. We refer to these cases as strict visual ambiguity. A slightly less strict form of visual ambiguity occurs in cases where attributes of two different primitives have different names, but the same values and constraints.

Another kind of visual ambiguity occurs when two primitive types or attribute values appear very similar, though not identical. Primitive types or attributes with a similar appearance may be visually indistinguishable, depending in part on the degree of similarity and the visual acuity of the viewer. For example, if two polygons of 15 and 17 sides are mapped to two different values of a data attribute, Figure 5 shows that users might not be able to correctly distinguish between the two values. Another example would be two primitive types with different attributes and values but the same appearance to the user.

Visual ambiguity is, in general, undesirable, because it results in schemas of a visual model that may not be correctly interpreted by a user. Strict visual ambiguity is straight-

forward to detect by testing the $\mathcal{Q}_{\mathcal{G}}$ and $\mathcal{R}_{\mathcal{G}}$ functions, and the constraints affecting the primitives in question. Non-strict ambiguity is much harder to define formally, let alone detect; there may not exist universal similarity (as opposed to equality) measures. Investigating possible definitions and similarity detection algorithms is part of our future work.

4.2.3. Semantic Ambiguity

Semantic ambiguity occurs when the appearance of a visual attribute value does not bring to mind the data attribute value with which it corresponds. The degree of ambiguity depends upon the memory of the user, the range of data attribute values, the type of visual attribute, and the choice of visual attribute values. For example, using randomly assigned colors to represent values between 1 and 500 would be problematic for any user lacking an eidetic memory (if the the values are of interest to the user). Using colors ordered and spaced by their place in the spectrum would be better, giving the user a feel for the magnitude of different values. Using Arabic numerals to represent these values would be the most precise (though possibly less good for giving a quick impression of the value). While improving human memory is beyond the scope of this paper, we can offer visual attribute and value choice guidelines to reduce semantic ambiguity.

In general, any visual attribute type can be used for representing a data attribute with a small value range. For example, it would not be difficult for a user to learn associations between a small number of shapes, colors, or patterns, and their corresponding data model values. Precisely capturing values with a larger range, however, requires visual attributes with inherent meaning; the visual value must in some way suggest the data value. Attribute types may be divided into two categories with respect to inherent meaning. Text and pictures can have much inherent meaning as long as the viewer shares a linguistic or cultural context with the creator. For example, a picture formed of a red octagon with the word "STOP" in the middle has an immediate association for most people. Shape, color, pattern, size, and location have less inherent meaning, and what meaning they have is limited to more narrow contexts. For example, a red colored light means "stop" in the context of driving, but it also means "on" in the context of electric kitchen ovens and toasters. If a precise representation is not needed, the limited inherent meaning of these attributes can be useful. For example, consider a metaphor that associates the spectrum of colors to a large range of temperatures. While specific values would be hard to determine, the user could easily make comparisons and determine general magnitudes of values.

As a result, in most cases text and pictures should be used to represent values with large ranges. In some cases, when a general impression of the value is needed instead of the exact value, other attributes can be used. Values with a smaller range, such as entity-class kind from Section 2.2, may be represented by any type of attribute.

5. Combining and Mixing Metaphors

Different metaphors have different characteristics: emphasis, space efficiency, intuitiveness, and versatility. There is no single metaphor that is best for all schemas and all situations. We believe that a schema visualization tool should support a variety of metaphors and associated visual models. Users will be able to choose among them so that the same data schema may be viewed as different visual schemas, each suitable for different circumstances.

The use of different metaphors may be taken one step further, by allowing the use of different metaphor correspondences for different parts of the same visual schema. This is faithful to the definition of metaphors in Section 3.1, which allows correspondences between one data model primitive type and several visual model primitive types. Such metaphors may originally be defined this way, or may be defined as a combination of two simpler metaphors. This involves combining their visual models into a single, unified model, and combining the metaphors to map from that model. The following section presents some example metaphors that will be used to demonstrate the formal specification of metaphor combination.

5.1. *Example Visual Metaphors*

Consider the data model $\mathcal{D}$ of *entity-classes* and *relationships* described in Section 2.4. Also consider the visual models, $\mathcal{G}_1$ and $\mathcal{G}_2$, described in the following table.

Visual model $\mathcal{G}_1$ is similar to $\mathcal{G}$ described in Section 2.4. Visual model $\mathcal{G}_2$ has *nodes* that are rectangles, and instead of using *edges* to represent a connection between two *nodes*, it uses *arrangements*. *Arrangements* are formed from a *text-display* construct and two *nodes*, a parent-node and child-node. The existence of an *arrangement* affects the location of the child-node. The specific physical arrangement is defined by a set of constraints which require *node* placement similar to a textual outline, where subpoints appear below and indented to the right of the main points. It should be noted that $\mathcal{G}_2$ is a visual model not for general directed graphs but only for trees, as any child *node* with multiple parents would have conflicting constraints on its location. These models are accompanied by several composition constraints, which are of no particular interest and are therefore not shown.

For each visual model, we define a metaphor. The two metaphors are shown in the table below. For brevity, the part of the metaphor that corresponds to the T_v function is not included. Visual metaphor $T_1 : \mathcal{G}_1 \rightarrow \mathcal{D}$ is similar to the metaphor described in Section 3.1, except that it does not include the *blob* primitive type, and *entity-class.kind* is represented by *node.label-color*. Visual metaphor $T_2 : \mathcal{G}_2 \rightarrow \mathcal{D}$ is different in that *entity-class.kind* is captured by *node.color*, and that *relationships* are expressed as physical arrangements of the related *nodes* (as described earlier). Figure 6 gives examples of a simple schema displayed using each of the metaphors. This example, drawn from the Cupid simulation model (Ioannidis, Livny, and Haber, 1992), shows a case where the outline metaphor is more compact than the graph metaphor.

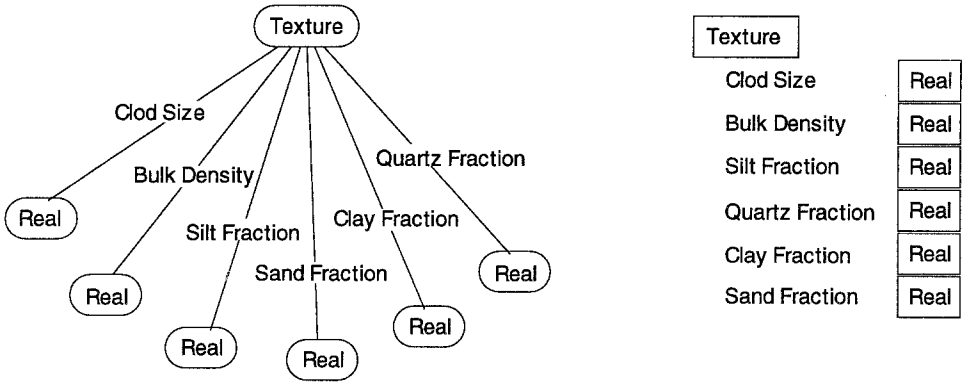

Figure 6. An example schema displayed using each of the two metaphors.

Model	Primitive Type (P)	Attribute ($P.A$)	Attribute Values ($\mathcal{R}^*(P.A)$)
$\mathcal{G}_1$	node	shape	{oval}
		location	plane-points
		size	{100 pixels}
		color	{white}
		label-text	text
		label-color	{blue,red}
	edge	source-location	plane-points
		dest-location	plane-points
		color	{red, orange, magenta, green}
		from-node	$\mathcal{I}$(node)
		to-node	$\mathcal{I}$(node)
		label-text	text
		label-color	{black}
$\mathcal{G}_2$	node	shape	{rectangle}
		location	plane-points
		size	{100 pixels}
		color	{yellow, brown}
		label-text	text
		label-color	{black}
	arrangement	label-text	text
		label-color	{red, orange, magenta, green}
		label-location	plane-points
		parent-node	$\mathcal{I}$(node)
		child-node	$\mathcal{I}$(node)

$x \in \mathcal{G}_1$	$T_1(x)$
node	entity-class
node.label-text	entity-class.name
node.label-color	entity-class.kind
edge	relationship
edge.label-text	relationship.name
edge.from-node	relationship.from-class
edge.to-node	relationship.to-class
edge.label-color	relationship.card-ratio

$x \in \mathcal{G}_2$	$T_2(x)$
node	entity-class
node.label-text	entity-class.name
node.color	entity-class.kind
arrangement	relationship
arrangement.label-text	relationship.name
arrangement.parent-node	relationship.from-class
arrangement.child-node	relationship.to-class
arrangement.label-color	relationship.card-ratio

5.2. *Combining Visual Models and Metaphors*

In order to use different metaphors for different parts of a schema, the visual models associated with these metaphors must be combined into a single model. In addition, the metaphors themselves must be combined to form a unified metaphor, mapping from the combined visual model to the data model. Combining the visual models ensures that the primitive types from different models may be used together, and that the metaphors themselves may be combined.

There are several abstractions that could be used to model the combination of visual models and metaphors, any of which results in a valid, unambiguous, and usable metaphor and schema mappings. These abstractions differ in the level of mixing that they permit of the visual models and metaphor functions. In this subsection, we discuss an abstraction that allows mixing at all levels. The subsequent subsections describe correctness and quality issues involved in mixing metaphors based on that abstraction.

For the visual models $\mathcal{G}_1 =< \mathcal{P}_{\mathcal{G}_1}, \mathcal{A}_{\mathcal{G}_1}, \mathcal{V}_{\mathcal{G}_1}, \mathcal{Q}_{\mathcal{G}_1}, \mathcal{R}_{\mathcal{G}_1}, \mathcal{C}_{\mathcal{G}_1} >$ and $\mathcal{G}_2 =< \mathcal{P}_{\mathcal{G}_2}, \mathcal{A}_{\mathcal{G}_2}, \mathcal{V}_{\mathcal{G}_2}, \mathcal{Q}_{\mathcal{G}_2}, \mathcal{R}_{\mathcal{G}_2}, \mathcal{C}_{\mathcal{G}_2} >$, consider their combination $\mathcal{G} =< \mathcal{P}_{\mathcal{G}}, \mathcal{A}_{\mathcal{G}}, \mathcal{V}_{\mathcal{G}}, \mathcal{Q}_{\mathcal{G}}, \mathcal{R}_{\mathcal{G}}, \mathcal{C}_{\mathcal{G}} >$. By definition, for any primitive type P that is common to both visual models, the equality $\mathcal{Q}_{\mathcal{G}_\infty}(P) = \mathcal{Q}_{\mathcal{G}_2}(P)$ holds, i.e., the same primitive type has the same attributes in all visual models that include it. The elements of the visual models

are combined as follows:

$$\begin{aligned}\mathcal{P}_{\mathcal{G}} &= \mathcal{P}_{\mathcal{G}_1} \cup \mathcal{P}_{\mathcal{G}_2} \\ \mathcal{A}_{\mathcal{G}} &= \mathcal{A}_{\mathcal{G}_1} \cup \mathcal{A}_{\mathcal{G}_2} \\ \mathcal{V}_{\mathcal{G}} &= \mathcal{V}_{\mathcal{G}_1} \cup \mathcal{V}_{\mathcal{G}_2} \\ \mathcal{C}_{\mathcal{G}} &= \mathcal{C}_{\mathcal{G}_1} \cup \mathcal{C}_{\mathcal{G}_2}.\end{aligned}$$

Note that, based on the naming convention established in Definition 1, the above equations imply that:

$$\begin{aligned}\forall P \in \mathcal{P}_{\mathcal{G}},\ \ \mathcal{Q}_{\mathcal{G}}(P) &= \mathcal{Q}_{\mathcal{G}_1}(P) \cup \mathcal{Q}_{\mathcal{G}_2}(P) \\ \forall P.A \in \mathcal{A}_{\mathcal{G}},\ \ \mathcal{R}_{\mathcal{G}}(P.A) &= \mathcal{R}_{\mathcal{G}_1}(P.A) \cup \mathcal{R}_{\mathcal{G}_2}(P.A).\end{aligned}$$

Given a combined visual model, two metaphors $T_1 = T_{p1} \cup T_{a1} \cup T_{v1}$ and $T_2 = T_{p2} \cup T_{a2} \cup T_{v2}$ may be combined to form a unified metaphor $T = T_p \cup T_a \cup T_v$ where

$$\begin{aligned}T_p &= T_{p1} \cup T_{p2} \\ T_a &= T_{a1} \cup T_{a2} \\ T_v &= T_{v1} \cup T_{v2}.\end{aligned}$$

5.3. *Correct and Good Mixing of Metaphors*

The result of combining two metaphors using the process shown in the previous section must satisfy Definition 3 in order for it to be a metaphor itself. Assume that T_1 and T_2 are correct metaphors with respect to either viewing or updating data schemas. Then, T_1 and T_2 are onto functions with possible additional properties of totality and/or 1-1ness. When taking the union of T_{x1} and T_{x2} (for $x \in \{p, a, v\}$), totality and onto-ness can never be lost. 1-1ness may be lost when unioning, but it is unrelated to correctness and only effects redundancy and choice in a metaphor (Section 3.2), so it does not present a problem. Functionality, however, is necessary for correctness and may be lost when unioning. In that case, T is not a correct metaphor, implying that the original metaphors are not combinable. To correctly combine T_1 and T_2, the resulting T_p, T_a, and T_v must be functions.

The combined metaphor must also satisfy the criteria from Section 4.1. In addition, the new set of constraints established by unioning the constraints of the two original visual models must contain no contradictions and should not exclude any visual schema that was valid in the two original visual models. If any of the above does not hold, then the original metaphors are not combinable.

We should emphasize once again that one could use a different abstraction from that described in Section 5.2 to combine metaphors. Such an abstraction would possibly allow different pairs of metaphors to become combinable. We have chosen the above abstraction for its simplicity and because it captures several desirable metaphor combinations.

5.4. *Example Metaphor Combination*

Consider the example metaphors from the previous section. When the two are combined, the metaphor will appear as follows:

x	$T(x)$
node	entity-class
node.label-text	entity-class.name
node.label-color	entity-class.kind
node.color	entity-class.kind
edge	relationship
arrangement	relationship
edge.label-text	relationship.name
arrangement.label-text	relationship.name
edge.from-node	relationship.from-class
arrangement.parent-node	relationship.from-class
edge.to-node	relationship.to-class
arrangement.child-node	relationship.to-class
edge.label-color	relationship.card-ratio
arrangement.label-color	relationship.card-ratio

Where both original metaphors are the same, such as the mapping of *entity-class* or *entity-class.name*, the combined metaphor is the same. Where the original metaphors diverge, the combined metaphor either offers choice (as in the case of *relationships*), or redundancy (as with *entity-class.kind*).

The unified visual model undergoes similar changes:

Primitive Type (P)	Attribute ($P.A$)	Attribute Values ($\mathcal{R}(P.A)$)
node	shape	{oval, rectangle}
	location	plane-points
	size	{100 pixels}
	color	{yellow, brown, white}
	label-text	text
	label-color	{blue,red, black}
edge	source-location	plane-points
	dest-location	plane-points
	color	{red, orange, magenta, green}
	from-node	$\mathcal{I}$(node)
	to-node	$\mathcal{I}$(node)
	label-text	text
	label-color	{black}
arrangement	label	text
	label-color	{red, orange, magenta, green}
	label-location	plane-points
	parent-node	$\mathcal{I}$(node)
	child-node	$\mathcal{I}$(node)

Nodes existed in both of the original models, yet they had different shapes. In the combined model, a choice of shape exists. Figure 7 gives an example of a schema displayed using the mixed metaphor. Note how *node* shape is either oval or rectangular, and how *relationships* may be displayed either using an *edge* or an *arrangement.*

6. Visually Capturing Additional Information

The definition of models and metaphors allows the visual model to have greater information capacity than the data model. Specifically, there may be more primitive types in the visual model than in the data model, the visual model primitive types used in the metaphor may have more attributes than their corresponding data model primitive types, and the range of visual attribute values may be greater than that of their corresponding data model attributes. It is possible to define the visual model to have the same information capacity as the data model, but often extra information capacity is valuable. Surplus information capacity may be used in two ways. One is to enrich the metaphor. For example, the T_p, T_a and T_v functions may be many-to-one, allowing redundancy and choice in representing information. The other use of extra information capacity, and the subject of this section, is to capture information outside of the data schema. This information may be divided into two categories: presentation and personal data model information. Presentation information has no meaning and simply improves the aesthetics of the visual schema. The *personal data model* is a superset of the data model, additionally containing information that is part of the user's conception but not

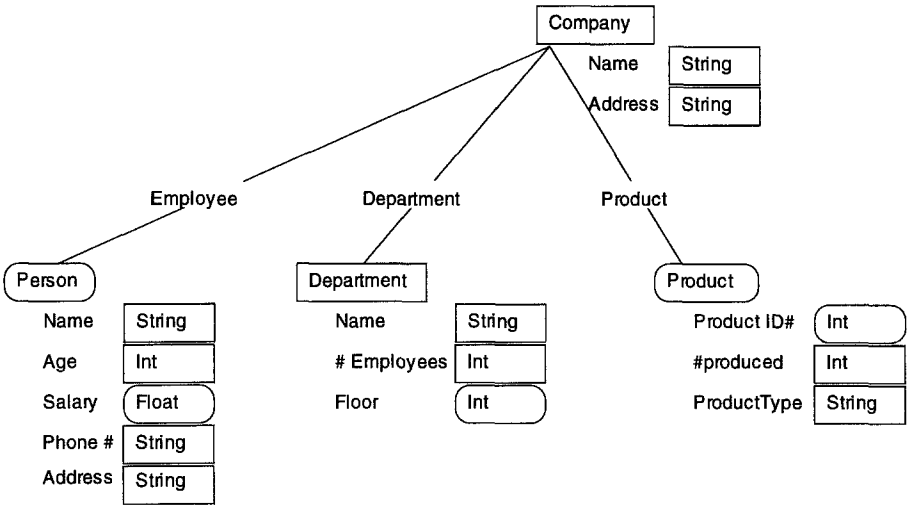

Figure 7. An example of a schema displayed using the combined metaphor.

captured by the database. These two kinds of information will be discussed in depth in this section.

6.1. Presentation Information

Presentation information is visual information that conveys no meaning to the user. For example, the locations of nodes in a directed graph could be chosen for purely aesthetic reasons. While not capturing any part of the data schema, this information is important as it can affect the readability of a presentation. There are many ways to lay out a directed graph, all having the same meaning, but some are much more readable than others.

Presentation information is captured by those surplus visual model primitives types and attributes that are not in the domain of the metaphor, and by the attribute value and primitive type choices that are part of the metaphor. For an example of the use of extra primitive types, consider the visual model and metaphor from Section 3.1 supplemented with the following primitive type:

Primitive Type (P)	Attribute ($P.A$)	Attribute Values ($\mathcal{R}^*(P.A)$)
rect	shape	{ rectangle }
	location	plane-points
	size	integer × integer pixels
	background-color	{ white }
	border-color	{ black }
	border-width	{ 2 } pixels

This *rect* is a black bordered rectangle of any given size and location. Not part of the metaphor's domain, it can be used freely without affecting the meaning of a visual schema. For example, a *rect* could be placed around the entire schema to give it a border and improve its appearance.

An example of surplus attributes may be found in the same metaphor: *node-location* is not specified by the mapping and can be freely specified as mentioned above. Similarly, the possibility of representing 1:1 relationships as either 'green' or 'orange' *edges* allows the user aesthetic leeway without changing the meaning of the visual schema.

Another means of capturing presentation information is through choice of visual model primitive types. Different primitive types may have very different appearances, affecting the aesthetics of the schema. For example, consider the combined metaphor described in Section 5.4. It maps two visual primitive types, *edge* and *arrangement*, to *relationships*. These visual primitive types have very different appearances and would be suitable in different situations.

6.2. *Personal Data Model Information*

Databases are commonly used for holding information about real-world items. Data schemas describe the organization of these items in as much detail as allowed by the data model. Frequently, however, there exists other organizational information about these items that might be helpful to the user, but is not or cannot be captured by the data schema.

We introduce the notion of a *personal data model* to capture the organization of the database from the user's viewpoint. This is no different from any other data model, except for the fact that each user may have a different personal data model (while there is a single system data model) and also that each personal data model must be an extension of the system data model. Accordingly, *personal visual metaphors* may be defined from a visual model to personal data models to capture the full range of characteristics of personal data schemas, as shown in Figure 8. Such an extended metaphor would map visual model primitive types, attributes, and values not used by the regular metaphor (i.e., its excess information capacity) to corresponding constructs of the personal data model that are not part of the system data model.

As an example of a personal data model and its corresponding personal metaphor, consider the data model of Section 2.4, whose primitive types are *entity-class* and *re-*

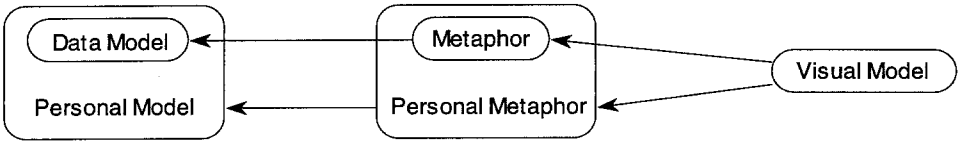

Figure 8. The Personal Data Model and Metaphor as Extensions of the Data Model and Visual Metaphor

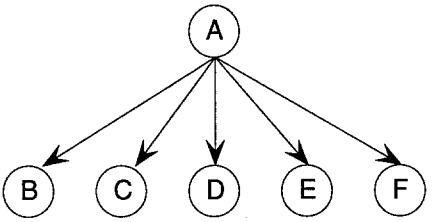

Figure 9. A directed graph.

lationship. Like other object-oriented/semantic data models, this model does not allow higher-level groupings of *entity-classes* or *relationships*, an ability that could be very useful to a user. For example, a schema combining data from several experiments could have *entity-classes* grouped by their original experiment, their roles within the experiment (e.g., input versus output), whether their contents are considered accurate, and their significance to the user. Multiple simultaneous orthogonal groupings may be captured this way, with an *entity-class* belonging to several different groups for different reasons. This may be achieved by allowing a user to extend the above data model so that groups can be captured. The visual metaphor from Section 3.1 may also be modified to represent group information to the user. In the original metaphor, *node* size, label-color, shape, and location are attributes that are not part of the metaphor mapping. If the visual model were defined to allow these attributes a greater range of values, they could be used by the personal metaphor to enhance the information that is captured visually. Figures 9, 10, and 11 give examples of an unmodified visual schema, grouping by location, and grouping by shape and location, respectively.

7. Related Work

Visual presentation of abstract information has been studied for more than 40,000 years, from pigments on cave walls to ink on paper to phosphor on the inside of video tubes. In its broadest sense, this field includes work in art, psychology, cognitive science, human-factors engineering, and many branches of computer science. The majority of this work

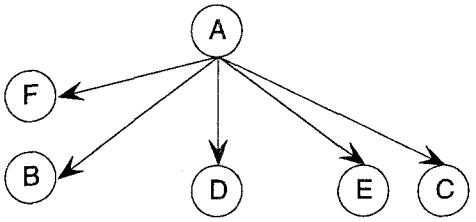

Figure 10. Grouping by location.

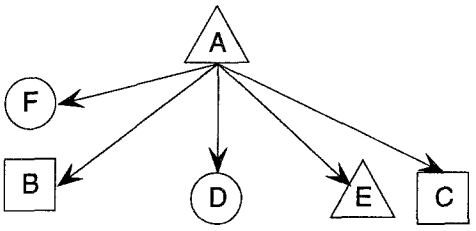

Figure 11. Grouping by location and shape.

deals with the evaluation of visualizations with respect to human perception (exemplified by the work of Edward R. Tufte (Tufte, 1983; Tufte, 1990). A much smaller body of work is concerned with the process of creating visualizations from abstract information; this work is concentrated in computer science due to its need to communicate computer data to humans. It is this area that is most closely related to our work.

There exist a large number of computer systems related to visualization. These include visualization tools such as DBMS GUIs, which display data and schemas, and computer assisted software engineering (CASE) tools, which create visualizations of data structures and program execution. On another level of abstraction are user interface tools, which allow users to create visualization tools. All of these systems are either explicitly or implicitly based on some conception of the process of visualization. Our formalism describes processes of visualization. As such it can be used as a means to classify and compare these other systems, and explain some of their resulting characteristics.

Our formalism identifies three distinct parts in the process of visualization:

- The data involved (the data model),
- The visualization (the visual model), and
- A transformation between the data and visualization (the metaphor).

The separation into three declarative descriptions permits metaphors to be evaluated, compared, and combined. It also allows personal data model and presentation information to be dealt with separately from visual information dependent on the metaphor.

In the following subsections, we use our formalism to evaluate visualization tools and user interface tools. In these descriptions, we focus on two important aspects of these systems: 1) how the models and metaphors are defined, and 2) whether or not it is possible to define or change the models and metaphors. These have an impact on the ability to test visualizations for correctness, to combine metaphors, and to supplement a visualization with personal data model and presentation information.

7.1. Visualization Tools

There are many computer tools for visualizing information, more than could be adequately covered here. We will discuss two areas of visualization tools that deal with structured information suitable to the models of our formalism. These are DBMS GUIs, and CASE tools.

7.1.1. DBMS GUIs

All database systems have some means to present schemas and data, though often the visual model is textual. Many systems do support GUIs with more advanced presentations; their metaphors may be broken down into the following categories:

- Tables, which use rows and columns to indicate database structure, as with QBE (Zloof, 1975) and other systems (Heiler and Rosenthal, 1985; Kuntz and Melchert, 1989; Ozsoyoglu, Matos, and Ozsoyoglu, 1989).
- Forms, which lay out information using a template that indicates structure, such as (King and Novak, 1987) and most commercial database systems.
- Diagrammatic presentations, such as E-R-like Diagrams (Angelaccio, Catarci, and Santucci, 1990; Elmasri and Larson, 1985; Leong, Sam, and Narasimhalu, 1989; Miura, 1991; Siau, Chan, and Tan, 1991; Wong and Kou, 1982) and other directed and non-directed graphs (Bryce and Hull, 1986; Consens and Mendelzon, 1990; Creasy, 1989; Gupta, Weymouth, and Jain, 1991; King and Melville, 1984; Lam, et al., 1990; Paredaens and Van den Bussche, 1992; Batini, et. al., 1991; Yoon, et al., 1987).
- Icons (pictures) that represent a concept or action (Catarci, Constabile, and Levialdi, 1991; Kaneko and Hara, 1986; Tsuda, et al., 1990).

For visualizing schemas, all of these systems have a fixed, hard-coded data model, visual model and metaphor. Although diagrammatic representations seem to be the most popular, the persistence of other approaches indicates that there is no single best metaphor. This further demonstrates the importance of flexible schema visualization. These systems

lack flexibility; they offer no choice in visualization, and no justification of their visual models and metaphors.

7.1.2. CASE Tools

CASE tools are environments to aid software development. One feature they usually provide is visualization of program data structures and execution. These visualizations can help the user to better understand the operation of the software. In most cases the data model, visual model, and metaphor are fixed. Evaluation, comparison, and combination of metaphors is not possible. A few of these systems, such as Incense (Myers, 1983), allow user definition of the visual model and metaphor, with alternate metaphors possible for a given data primitive type. These are defined procedurally, however, so determination of metaphor correctness is not possible.

7.2. User-Interface Tools

User Interface tools assist programmers in creating graphical user interfaces. They provide frameworks through which users can specify the appearance and interaction characteristics of an interface. Part of the specified interface may be a visualization of information; as such these systems support the description of data visualizations. We consider three kinds of user interface tools: general tools, tools built on top of DBMSs, and automatic tools.

7.2.1. General User Interface Tools

General user interface tools allow the user to create a visual model for a visualization, though they use a variety of means to describe it. For example, the Chiron-1 (Taylor and Johnson, 1993) and Motif (OSF, 1990) systems rely upon procedural specifications, where the user describes the structure of the visualization through calls to a library. Marquise (Myers, McDaniel, and Kosbie, 1993) is an example of a demonstrational system, where the user specifies the appearance and behavior of an interface by drawing and laying out objects on the screen. InterViews (Linton, Calder, and Vlissides, 1988) is similar, allowing the user to specify interface appearance by laying out special objects in a graphical editor. Behavior of the interface, however, must be specified procedurally in InterViews. HUMANOID (Szekely, Luo, and Neches, 1993) and UIDE (Sukavariya, Foley, and Griffith, 1993) allow description of data and visual models through expressions in formal modeling languages.

Most of these systems also allow specification of a metaphor, though in some cases the metaphor is closely tied to the visual model. HUMANOID embeds in each visual model primitive type a procedural description of the data to be presented. Lower level toolkits such as Motif and InterViews require procedural specifications of all parts of the data model and metaphor. Chiron also specifies metaphors (called "artists") procedurally,

though they are separate from the visual model. All of these systems allow use of different metaphors (e.g., the InterViews package was used to create our tool based on the formalism). UIDE does not support a metaphor as described in our formalism. It defines correspondences between visual and data model primitives, but not between attributes or values. Instead it establishes correspondences between actions on visual model objects (such as a click of the mouse) and actions on data model objects (such as a change in a value). Marquise does not support a data model as distinct from the visual model, and as such does not need metaphors. None of these systems allows declarative definition of metaphors. As a result, they cannot test metaphors or metaphor combinations for correctness, nor can they evaluate or compare metaphors.

7.2.2. DBMS User Interface Tools

A related area is DBMS User Interface tools. These include O2Look/ToonMaker (Borras, et al., 1992), ODDS (Flynn and Maier, 1992), FaceKit (King and Novak, 1989), and Picasso (Rowe, et al., 1990). These tools are oriented toward building interfaces, but unlike other interface toolkits, they also interact with a database explicitly, using it to store interface information and simplifying the specification of visualizations for database objects. A related system is DOODLE (Cruz, 1992), which provides a visual language for querying an OODBMS and defining visualizations of database objects.

These systems use the data model of the underlying database as the data model. Each allows definition of the visual model in a different manner: ToonMaker provides an interactive visual editor for creating visual primitives, ODDS uses declarative descriptions, FaceKit procedurally specifies visualizations in methods of the object class to be displayed, and Picasso uses widgets defined in Lisp. These systems do allow different visualizations for any data object. As with many of the user interface tools described above, however, these systems do not represent metaphors as separate from visualizations; the definition of a visual item is tied to the database item it is to represent. Lacking a separate metaphor, these systems do not evaluate or compare visualizations, or test metaphor combinations for consistency.

7.2.3. Automatic User Interface Tools

Some user interface tools automatically generate presentations from a description of the data (DON (Kim and Foley, 1993), Dost (Dewan and Solomon, 1990), GENIUS (Janssen, Weisbecker, and Ziegler, 1993), TRIDENT (Vanderdonckt and Bodart, 1993), using a set of predefined rules to determine presentation and interaction. These rules are analogous to metaphors in that they describe the mapping from data to visualization. The rules are hard-wired, however, so they cannot be changed or combined. Thus, there is no determination of metaphor correctness, no possibility of mixing, and no flexibility in the use of non-data model information.

7.3. Other Formalism Work

The work of Kuhn and Frank (Kuhn and Frank, 1991) is related to our work, yet covers a different area. It uses algebraic mappings to study the behavior of user interfaces. For example, it considers the similarities and differences between operations on a physical desktop, and those on a computer's virtual desktop. This permits evaluation of correctness of the behavioral aspects of visual metaphors. An earlier paper on the same subject by these authors (Kuhn, Jackson, and Frank, 1991) encouraged some of our early ideas that grew into our formalism.

8. Conclusions and Future Work

In this paper, we have presented a formalism for visual metaphors and described how it may be used to improve the visual presentation of data schemas. This formalism allows high level description of the correspondence between data and visual models. This description allows simpler definition of metaphors, easier evaluation and comparison of metaphors, and combination of different metaphors. The formalism can help improve schema visualizations in the many roles they play.

Currently, a large part of the formalism has been implemented as a schema editing tool. Arbitrary data models, visual models, and metaphors between them can be defined, although in a hard-wired manner. (Our future work includes enhancing the interface to allow users to define all of these dynamically.) The user can create and modify visual schemas using direct-manipulation tools; these visual schemas are translated to the appropriate data schemas based on the metaphor. We have tested the system with various metaphors and data models and it works well, though limited space precludes a more lengthy description of the schema editing tool in this paper.

Future work includes completing the schema editing tool. Two areas require effort: allowing user definition of different visual models and metaphors, and creating a sufficiently expressive language for specifying all necessary constraints. In addition, the formalism should be examined for solutions to the problems of displaying very large schemas, which are common in scientific databases (where schemas with thousands of classes and tens of thousands of relationships are common). The various formalism criteria for distinguishing metaphors need to be examined for testability; some may be undecidable (e.g., visual ambiguity). Furthermore, the formalism should be expanded to include "views", visualizations that contain subsets of data model information. It is also important to determine applicability of the formalism to data models beyond the object-oriented model with which we work; databases based upon other current models, and earlier models in legacy systems (e.g., the network or un-normalized relational models) would also benefit from improved schema visualization. Finally, once the formalism and the system based on it are sufficiently developed, it will be important to evaluate them empirically. Such an experiment would demonstrate the advantages and disadvantages of our formal approach to visualization.

Acknowledgements

Dr. Ioannidis' work has been partially supported by the National Science Foundation under Grants IRI-9113736, IRI-9224741, and IRI-9157368 (PYI Award) and by grants from DEC, IBM, HP, AT&T, and Informix. Dr. Livny's work has been partially supported by the National Science Foundation under Grant IRI-9224741. Special thanks to Janet Wiener and Renee Miller for their proofreading, without whom this paper would have been more dense and less clear.

Notes

1. The term *metaphor* is sometimes used to describe behavior of visualizations as well as appearance. We do not consider behavior in this paper.
2. For simplicity, we use the names of attributes directly instead of their corresponding full identifiers, i.e., A instead of $P.A$. This also holds for all other models presented in this paper and applies to any constraints that are shown as well.
3. Note that if P does not have an image under T_p, then $\mathcal{Q}_{\mathcal{D}}(T_p(P))$ is the empty set. Therefore T_a^P is empty as well which makes it vacuously an onto function. Similar observations hold for $T_v^{P.A}$.
4. Non-onto functions and non-functional correspondences will prove useful in extending metaphors to allow visualizations of subsets of data schema information, part of our future work.

References

M. Angelaccio, T. Catarci, and G. Santucci. QDB*: A Graphical Query Language with Recursion. *IEEE Transactions on Software Engineering*, pages 1150–1163, October 1990.

R. Agrawal, N.H. Gehani, and J. Srinivasan. OdeView: The Graphical Interface to Ode. *SIGMOD Record*, pages 34–43, 1990.

C. Batini, T. Catarci, M.F. Costabile, and S. Levialdi. Visual Query Systems, A Taxonomy. In *IFIP Conference on Visual Database Systems*, 1991.

D. Bryce and R. Hull. SNAP: A Graphics-based Schema Manager. In *Proceedings of the International Conference on Data Engineering*, pages 151–164, 1986.

A. J. Benjamin and K. M. Lew. A Visual Tool for Managing Relational Databases. In *Proceedings of the International Conference on Data Engineering*, 1986.

P. Borras, J.C. Mamou, D. Plateau, B. Poyet, and D. Tallot. Building user interface for database applications. *SIGMOD Record*, pages 32–38, March 1992.

T. Catarci, M.F. Costabile, and S. Levialdi. Iconic and Diagramatic Interfaces, an Integrated Approach. In *IEEE Workshop on Visual Languages*, 1991.

M. Consens and A. Mendelzon. GraphLog: a Visual Formalism for Real Life Recursion. In T.I. Kunnii, editor, *Visual Database Langauges*, pages 404–415. North-Holland, Amsterdam, The Netherlands, 1990.

P. Creasy. ENIAM: A More Complete Conceptual Schema Language. In *Proceedings of the International Conference on Very Large Data Bases*, pages 107–114, 1989.

I. F. Cruz. DOODLE: A Visual Language for Object-Oriented Databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 71–80, June 1992.

P. Dewan and M. Solomon. An Approach to Support Automatic Generation of User Interfaces. *ACM Transactions on Programming Languages and Systems*, 12(4):566–609, October 1990.

M. Egenhofer and A. Frank. Towards a Spatial Query Language: User Interface Considerations. In *Proceedings of the International Conference on Very Large Data Bases*, pages p124 – 133, 1988.

R. Elmasri and J. Larson. A Graphical Query Facility for ER Database. In *Proceedings of the 1985 IEEE Conference on the E-R Approach*, pages 236–245, 1985.

B. Flynn and D. Maier. Supporting Display Generation for Complex Database Objects. *SIGMOD Record*, pages 18–24, March 1992.

J. D. Foley, A. van Dam, S. K. Feiner, and J.F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, Reading, Massachusetts, 1990.

A. Gupta, T. Weymouth, and R. Jain. Semantic Queries with Pictures: The VIMSYS Model. In *Proceedings of the International Conference on Very Large Data Bases*, pages 69–79, 1991.

S. Heiler and A. Rosenthal. G-Whiz*, a Visual Interface for the Functional Model with Recursion. In *Proceedings of the International Conference on Very Large Data Bases*, pages 209–218, 1985.

Y. Ioannidis and M. Livny. Data Model Mapper Generators In Observation DBMSs. In *Heterogeneous DB Workshop*, December 1989.

Y. Ioannidis and M. Livny. Conceptual Schemas: Multi-Faceted Tools for Desktop Scientific Experiment Management. *International Journal of Intelligent and Cooperative Information Systems*, 1(3), December 1992.

Y. Ioannidis, M. Livny, and E. M. Haber. Graphical User Interfaces for the Management of Scientific Experiments and Data. *SIGMOD Record*, pages 47–53, March 1992.

Y. Ioannidis, M. Livny, E. M. Haber, R. Miller, O. Tsatalos, and J. Wiener. Desktop Experiment Management. *IEEE Data Engineering Bulletin*, 16(1):19–23, March 1993.

C. Janssen, A. Weisbecker, and J. Ziegler. Generating User Interfaces from Data Models and Dialogue Net Specifications. In *INTERCHI '93, Proceedings of the Conference on Human Factors in Computing Systems*, pages 419–423, April 1993.

W. Kuhn and A. U. Frank. A Formalization of Metaphors and Image-Schemas in User Interfaces. In H.M. Mark and A.U. Frank, editors, *Cognitive and Lunguistic Aspects of Geographic Space*, pages 419–434. Kluwer Academic Publisher, Amsterdam, The Netherlands, 1991.

W. C. Kim and J. D. Foley. Providing High-level Control and Expert Assistance in the User Interface Presentation Design. In *INTERCHI '93, Proceedings of the Conference on Human Factors in Computing Systems*, pages 430–437, April 1993.

A. Kaneko and Y. Hara. A Multimedia Document Base System for Office Work Support. In *IEEE COMPSAC*, pages 336–343, 1986.

W. Kuhn, J. P. Jackson, and A. U. Frank. Specifying Metaphors Algebraically. *SIG CHI Bulletin*, January 1991.

R. King and S. Melville. SKI: A Semantic-Knowlegeable Interface. In *Proceedings of the International Conference on Very Large Data Bases*, pages 30–33, 1984.

M. Kuntz and R. Melchert. Pasta-3's Graphical Query Language: Direct Manipulation, Cooperative Queries, Full Expressive Power. In *Proceedings of the International Conference on Very Large Data Bases*, pages 97–105, 1989.

R. King and M. Novak. Freeform: A User-Adaptable Form Management System. In *Proceedings of the International Conference on Very Large Data Bases*, pages 331–339, 1987.

R. King and M. Novak. FaceKit: A Database Interface Design Toolkit. In *Proceedings of the International Conference on Very Large Data Bases*, pages 115–123, 1989.

R. King and M. Novak. Building Reusable Data Representations with FaceKit. *SIGMOD Record*, pages 11–17, March 1992.

H. Lam, H. M. Chen, F. S. Ty, J. Qiu, and S.Y.W. Su. A Graphical Interface for an Object-Oriented Query Language. In *IEEE COMPSAC*, pages 231–235, 1990.

M. A. Linton, P. R. Calder, and J. M. Vlissides. InterViews: A C++ Graphical Interface Toolkit. Technical Report CSL-TR-88-358, Stanford University, July 1988.

M. Leong, S. Sam, and D. Narasimhalu. *Towards a Visual Language for an Object-Oriented Multi-Media Database System*, pages 465–495. Elsevier Science Publishers B.V., 1989.

R. Miller, Y. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and tran slation. In *Proc. 19th Int. VLDB Conference*, Dublin, Ireland, August 1993.

T. Miura. A Visual Data Manipulation Language for a Semantic Data Model. In *IEEE COMPSAC*, pages 212–218, 1991.

J.C. Mamou and C. B. Medeiros. Interactive Manipulation of Object-Oriented Views. In *Proceedings of the International Conference on Data Engineering*, pages 60–69, 1991.

B. A. Myers, R. G. McDaniel, and D. S. Kosbie. Marquise: Creating Complete User Interfaces by Demonstration. In *INTERCHI '93, Proceedings of the Conference on Human Factors in Computing Systems*, pages 293–300, April 1993.

D. Maier, P. Nordquist, and M. Grossman. Displaying Database Objects. Technical Report CS/E 86–001, Oregon Graduate Institute, January 1986.

A. Motro. BAROQUE - a Browser for Relational Database. *ACM TOIS*, 4(2):164–181, April 1986.

B. A. Myers. INCENSE: A System for Dispalying Data Structures. *Computer Graphics*, 17(3):115–125, July 1983.

G. Ozsoyglu, V. Matos, and Z. M. Ozsoyglu. Query Processing Techniques in the Summary-Table-by-Example Database Query Language. *ACM Transactions on Database Systems*, pages 526–573, December 1989.

Open Software Foundation, Englewood Cliffs, New Jersey. *OSF/Motif Style Guide, Revision 1.0*, 1990.

J. Paredaens and J. Van den Bussche. An Overview of GOOD. *SIGMOD Record*, pages 25–31, March 1992.

L. Rowe, J. Konstan, B. Simth, S.Seitz, and C. Lin. The Picasso Application Framework. Technical Report UCB/ERL M90/18, University of California, Berkeley, March 1990.

K.L. Siau, H.C. Chan, and K.P. Tan. Visual Knowledge Query Language as a Front-end to Relational Systems. In *IEEE COMPSAC*, pages 373–378, 1991.

P. 'Noi' Sukavariya, J. D. Foley, and T. Griffith. A Second Generation User Interface Design Environment: The Model and The Runtime Architecture. In *INTERCHI '93, Proceedings of the Conference on Human Factors in Computing Systems*, pages 375–382, April 1993.

M. Stonebraker and J. Kalash. TIMBER - A Sophisticated Relational Browser. In *Proceedings of the International Conference on Very Large Data Bases*, pages 1–10, 1982.

P. Szekely, P. Luo, and R. Neches. Beyond Intferface Builders: Model Based Interface Tools. In *INTERCHI '93, Proceedings of the Conference on Human Factors in Computing Systems*, pages 383–390, April 1993.

K. Tsuda, M. Hirakawa, M Tanaka, and T. Ichickawa. Iconic Browser: An Iconic Retrieval System for Object-Oriented Databases. *Journal of Visual Languages and Computing*, 1, 1990.

R.N. Taylor and G. F. Johnson. Separation of Concerns in the Chiron-1 User Interface Development and Management System. In *INTERCHI '93, Proceedings of the Conference on Human Factors in Computing Systems*, pages 367–374, April 1993.

E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Conn., 1983.

E. R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, Conn., 1990.

J. M. Vanderdonckt and F. Bodart. Encapsulating Knowledge For Intelligent Automatic Interaction Objects Selection. In *INTERCHI '93, Proceedings of the Conference on Human Factors in Computing Systems*, pages 424–429, April 1993.

H. K. T. Wong and I. Kou. GUIDE: Graphical User Interface for Database Exploration. In *Proceedings of the International Conference on Very Large Data Bases*, pages 22–32, 1982.

B. D. Yoon, B. Do, F. Suzuki, H. Ishikawa, and A. Makinouchi. Experimental Multimedia DBMS Ising an Object-Oriented Approach. In *IEEE COMPSAC*, pages 632–641, 1987.

M. Zloof. Query-by-Example, The Invocation and Definition of Tables and Forms. In *Proceedings of the International Conference on Very Large Data Bases*, 1975.