3

Building Metaphors for Supporting User Interaction with Multimedia Databases

M. F. Costabile, D. Malerba
Dipartimento di Informatica, Università degli Studi di Bari
Via Orabona 4, I-70126 Bari, Italy
{costabile, malerba}@di.uniba.it

M. Hemmje, A. Paradiso
Integration Information Systems Institute, GMD
Dolivostrasse 4, D-64293 Darmstadt, Germany
{hemmje, paradiso}@darmstadt.gmd.de

Abstract

In this paper we discuss how metaphors for supporting user interaction with multimedia databases can be automatically generated. The work presented is a further step in the development of Virgilio, a Virtual Reality (VR) based system that has been designed to be a general purpose exploration tool for highly structured data. Virgilio visualizes the results of a query to a database by generating VR scenes, that exploit appropriate metaphors in order to take advantage of common knowledge about real world objects, thus reducing the cognitive load in the process of information assimilation. We analyze two specific components of the Virgilio architecture, the Query Management Tool and the Metaphor Definition Tool, and we identify a completely automatic procedure to define the metaphor that will be exploited in the construction of the VR scene. The implementation of this procedure exploits the backtracking search strategy of Prolog interpreters to solve a typical constraint satisfaction problem.

Keywords

Multimedia Database, User Interaction , Metaphor, Constraint Satisfaction

Visual Database Systems 4 Y. Ioannidis & W. Klas (Eds.)

1 INTRODUCTION

In many of nowadays web-based environments for electronic marketing and commerce, that present large multimedia product and service catalogues, it becomes more and more difficult to provide naive end users, such as private consumers or commercial business partners, with intuitive user interfaces to access the large multimedia collections describing the presented products and services. The same holds for marketing managers and other employees responsible for managing and maintaining the large and constantly changing set of multimedia information chunks and fragments contained in these collections.
As a consequence, many efforts are devoted to improve the quality of the interaction between users and databases. Virtual Reality (VR) techniques are a promising interaction paradigm particularly suited to novice and/or occasional users. The users are facilitated in the database navigation since the system proposes them an environment that reproduces a real situation and gives the possibility of interacting by manipulating objects that have a direct correspondence with known objects.

VR techniques combine the advantages of 3D visualizations with the power of metaphorical representations. Presenting the result of a database query through a VR scene allows users to explore data more easily since they interact with familiar objects. The structural and dynamic properties of the objects in the virtual world, i.e. the way objects can be composed and can act themselves, are predictable since they belong to the users' general background. No particular training should be required to interact with and explore the dataset, thus reducing the learning overhead of naive users when accessing information.

Virgilio is a VR based system that has been designed to be a general purpose exploration tool for highly structured data. It is capable of visualizing large sets of objects of considerable intra-object and inter-object complexity through effective VR techniques. Virgilio is based on several metaphors in order to take advantage of common knowledge about real world objects, thus reducing the cognitive load in the process of information assimilation. The overall system has been presented in (Massari *et al.*, 1997), where lot of emphasis was posed on the generation of the 3D scenes, once the metaphor exploited in the visualization was chosen in a semi-automatic way, i.e. with the intervention of the system administrator. In this paper we discuss how metaphors for supporting user interaction with multimedia databases can be automatically generated in Virgilio. Therefore, we are more concerned with specific components of the Virgilio architecture, the Query Management Tool and the Metaphor Definition Tool, whose aim is to identify with a completely automatic procedure the most appropriate metaphor to be exploited in the construction of the VR scene.

Our work represents a further step in the Virgilio project, since we have automated the choice of a proper VR visualization of the results of the user's query. In other words, the definition of the mapping (or metaphor) between the

query result and objects of a virtual world, that was a task of the System Administrator in the previous release of Virgilio is now performed automatically by the system.

The content of the paper is the following. Section 2 introduces the concepts of logical, physical, and VR information spaces. Section 3 discusses metaphor in user interfaces and illustrates its use in the Virgilio system. The architecture of Virgilio is presented in Section 4. Section 5 describes the process that generates from a query to the database all information necessary to the construction of the VR scene. Section 6 gives the conclusions.

2 INFORMATION SPACES: FROM LOGICAL TO VR

In order to allow users easy access to a database, the information stored in the database needs to be visualized in an information space. This visualization can either be carried out by the user in the user's mind, in which case it is essentially the user's conceptualization of the database; or the visualization could be accomplished by the system, in which case the visualization is generated on the display screen. The latter is what it is actually defined information visualization, i.e. "a process of transforming information into a visual form enabling the user to observe information" (Gershon *et al.*, 1997). The essence of this process is to visually present information that is non inherently visual, such as text. Recent research has proved that successful visualization can reduce the time to get information, and to make sense out of it; it also enhance creative thinking.

Database objects, in general, are abstracted from real-life objects in the real world. Therefore, we can distinguish the logical information space and the physical information space (Chang and Costabile, 1997). In the logical space, the abstract database objects are represented. In the physical space, the abstract database objects are materialized and represented as physical objects that reflect real-life objects, such as diagrams, icons and sketches. For example, each object is materialized as an icon, and the physical information space consists of a collection of icons. These icons can be arranged spatially, so that the spatial locations approximately reflect the relations among database objects.

To create visualizations, the information in the logical space must be mapped into a physical space that will represent relationships contained in the information faithfully and efficiently. In this way, users will exploit their innate abilities to understand spatial relationships, also shifting most cognitive processing load to the perceptual system.

In the physical information space, the objects reflect real-world objects, but the world is still an abstract world. One further step is to present information in a VR information space. VR allows the users to be placed in a 3D environment they can directly manipulate. What the users see on the screen will be the same as what can be experienced in the real world. 3D features can be used to present the

results in a VR setting. For example, if the database refers to the books of a library, we can represent a Virtual Library in which the physical locations of books are indicated by blinking icons in a 3D presentation of the book stacks of the library. What the user sees on the screen will be the same (after simplifications) as what can be experienced in the real world.

It is worth noting that we are talking about "nonimmersive" VR (Robertson *et al.*, 1993), that is the user is placed in a 3D environment he/she can directly manipulate without wearing head-mounted stereo displays or special gloves, but acting only with mouse, keyboard, and monitor of a conventional workstation.

The real world, from which the database objects are abstracted, is the environment that the database objects must relate to. The real world is often abstracted in the information space. Only in the VR information space will the real world be represented in a direct way. Indeed, finding a good spatial representation of the information at hand is one of the most difficult tasks in visualization of abstract information. The key problem in information visualization is to invent visual metaphors for non physical data (Gershon *et al.*, 1997). Next section discusses the power of metaphorical representation and their use in database interaction.

3 METAPHOR IN USER INTERFACES

The literal meaning of metaphor (from the Greek word 'metaphorein') is to transfer or to carry across. One of the most important aspects of metaphor is that it gives the possibility of going from familiar concepts to unknown ones. The definition of metaphor given by Lakoff and Johnson (1980) says "*metaphor* is a rhetoric figure, whose essence is understanding and experiencing one kind of thing in terms of another". Thanks to the metaphor we can move from familiar concepts to unknown ones, thus incorporating new knowledge in old. Often, for introducing a new concept we present it in relation to a well known one, thus simplifying the learning process. For example, the model of the atom is usually presented with reference to the structure of the solar system.

Metaphors consist of two sets of component concepts, a *target* component and a *source* component (Martin, 1990). The target consists of the concepts we are actually referring to (also said the original idea). The source refers to the concepts in terms of which the intended target concepts are being viewed (the borrowed idea). Conventional metaphors are represented as sets of associations, or relations, between source and target concepts. Source and target concepts usually belong to different domains, and the familiarity with the source domain is exploited to understand the target concepts. The metaphor specifies how the source concepts correspond to the various target concepts. It establishes a mapping between target and source domains.

Metaphor is acknowledged as a fundamental tool in creative interface design, since it provides the user with a friendlier environment to work with (Mountford, 1990; Erikson, 1990). It is well known that an ideal metaphor does not exist, but it is extremely important to choose the metaphor which is appropriate depending on the particular situation. Some insights on metaphorical design are given in (Marcus, 1994; Madsen, 1994). In database interfaces, metaphors have been exploited for representing the intensional part of the database, that is the data schema; in such cases, the metaphor mediates between the data model and the user (Haber *et al.*, 1994; Catarci *et al.*, 1995). Most end users are actually concerned with the extensional part of the database, therefore it is appropriate to offer them a scenario where the information contained in the database is metaphorically represented in a VR environment, i.e. in a virtual world, so that the user is no longer aware of a presence of a structured database, but he/she is interacting as in the real world.

Virgilio is a system that supports the definition of a metaphor as a mapping between data in the result of a query to a database and objects in a virtual world. Several metaphors are actually available in the system, so that different mappings between a same data set, representing the target domain, and different virtual worlds, each representing a source domain, can be generated in order to present to users the most effective environment for their preferences and expectations. One of the most used virtual worlds in Virgilio is a "building" with several floors, and an entrance with an elevator to reach different floors. On each floor, there is a corridor with several rooms, and in each room there are pieces of furniture, as it happens in the real world. Different virtual worlds are available in Virgilio: one is a book store, where there are several areas, each one with several shelves on which books are shown; another is a ship, with the different elements that are typical in real ships. Yet, other virtual worlds can be input into the system. The main contribution of this paper is the automatic construction of the mappings between the dataset and the available virtual worlds, as described in Section 5.

4 VIRGILIO ARCHITECTURE IN BRIEF

The architecture of the overall Virgilio system, as described in (Massari et al., 1997), is shown in Figure 1. The main components are: a) three modules, called Query Management Tool, Metaphor Definition Tool, Virtual World Object Editor; b) a global repository of information that includes three meta databases containing information necessary to generate the visualizations, namely the Query Repository, the Metaphor Repository and the Virtual World (VW) Objects Repository; c) the Scene Constructor Server. The other items displayed in Figure 1, that is a DBMS with the database storing the data the users want to access to, a Web Browser, and an unspecified web network connection are considered external to Virgilio. The database is assumed to be a generic one with a structure

composed of different kinds of objects, many semantic relationships, and possibly containing multimedia data.

Virgilio has been designed to be a general purpose exploration environment for highly structured data. It is capable of visualizing large sets of objects of considerable intra-object and inter-object complexity through effective VR techniques. The data resulting from a query execution are presented in a 3D virtual environment by exploiting appropriate metaphors that refers to the various virtual worlds available in Virgilio.

Information on the virtual worlds and all objects they include is stored in the Virtual World Object Repository, and it is provided to the system through the Virtual World Object Editor.

The Query Repository stores representations of both the performed query and the data set that is the result of a query (called answer set in the rest of the paper), that are needed for the process of metaphor definition and the construction of the VR scene. By metaphor definition we mean the definition of the mapping between objects in the database and objects in a virtual world, that are chosen among those available in the system. This is done by the Metaphor Definition Tool, and information about such a mapping is stored in the Metaphor Repository. The VR scene is generated by the Scene Constructor Server on the basis of the information stored in the various repositories.

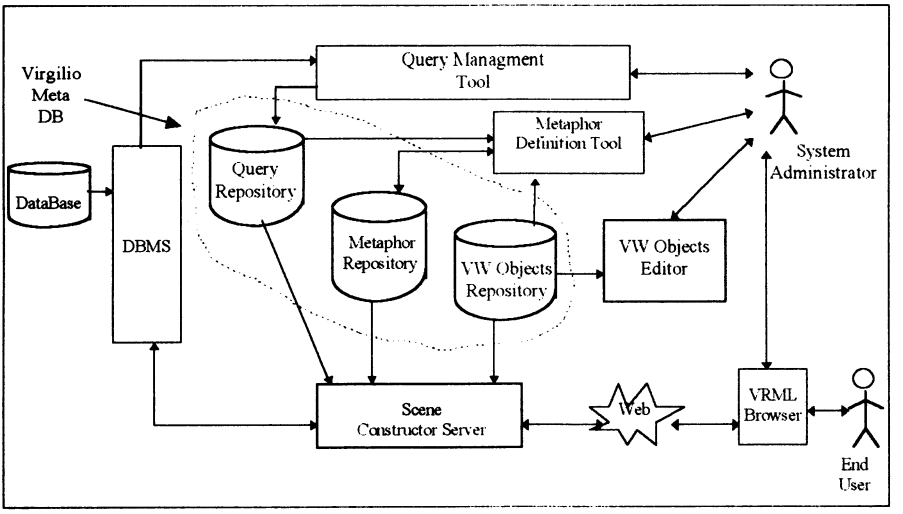

Figure 1. Architecture of Virgilio System.

As shown in Figure 1, there are two different types of users interacting with Virgilio: the end user and the system administrator. End users interact with Virgilio by retrieving 3D scenes and browsing embedded information by means of a VRML Browser. A typical interaction between Virgilio and the end user is:

the user starts browsing a VR scene; when the user decides to navigate another scene of a virtual world, a message is sent back to Virgilio which, in response, will generate a new scene that will be displayed on the screen, and so on.

In the architecture proposed in (Massari *et al.*, 1997), the system administrator was a fundamental intermediary between end user and system, since he/she performed three important tasks:

1. defining queries according to users' needs;
2. specifying a set of proper VR visualizations of such queries by defining a mapping (or metaphor) among data in the query results and objects of a virtual world;
3. defining new virtual world objects, specifying both their visual aspects and the containment relationships with other objects.

If we want Virgilio to become a complete VR based system for both querying a generic database and browsing the query results, we should automate the above first two tasks, so that no human intermediary should be necessary in the dialogue between the end user and the system. Task 3 is the only one that goes beyond the capabilities of a end user; defining a new virtual world by specifying all its objects with their attribute is a complicate task that needs to be done off-line by a team of design specialists. The new VR objects are input to the system through the Virtual world object editor, and stored into the VR object repository, so that they will be accessible by both the metaphor definition tool and the Scene Constructor Server.

In the next section we describe how we have automated task 2, so that once a query has been formulated by some visual query interface, translated in an appropriate language used by the DBMS and the query results are retrieved in the database, such results are suitably processed in order to automatically define the appropriate metaphor for their visualization in the VR scene.

5 FROM QUERY TO RESULT VISUALIZATION

We now describe the whole process that generates, from a query to a database, a VR scene that allows the user to browse the query results. The main steps of this process are the generation of a so-called structure tree and its Prolog representation (see below), the construction of a mapping from the structure tree into a virtual world, and the visual representation of such a mapping. The original query is actually input to VIRGILIO through a visual query interface, whose analysis goes beyond the scope of this paper (see (Catarci *et al.*, 1997) for examples of visual query interfaces). Such an interface translates a visual query into an SQL query which can be managed by a standard relational DBMS.

5.1 Generation of the structure tree

In the Virgilio operating framework, the DB model is richer than a simple relational one. Indeed, Virgilio operates with those models which can support the notion of nested relation (Atzeni and De Antonellis, 1993). Informally speaking, a nested relation is a set of tuples such that the values of attributes are allowed to be nested relations themselves. Nested relations organize information in a hierarchical structure, which may also be detected in the query results. In this context, we can state that Virgilio is a VR system for the visualization and exploration of nested relations according to the browsing paradigm.

Obviously, when Virgilio operates on extended relational DB the result of a query is a nested relation. However, when visual queries are transformed into standard SQL queries, the query result is a flat relation, which is not appropriate to be browsed. In this case it is possible to apply the nest operator that produces nested relations from flatter ones (Atzeni and De Antonellis, 1993). For instance, let us consider the following SQL query concerning a database of songs where also information about singers, published CD's, and music types is available together with some relationships among these entities.

Query 1:

```
SELECT musicType.name, musicType.notes, band.name, band.photo, album.title,
       album.cover, song.title
FROM   musicType, band, album, song , tipicSings, contained , published
WHERE  album.code=contained.albumCode AND tipicSings.bandCode=band.code
       AND album.code=published.albumCode AND band.code=published.bandCode
       AND song.code=contained.songCode
       AND tipicSings.musicName= musicType.name
ORDER BY musicType.name, band.name, album.title;
```

The result of the query would be a flat relation that can be transformed into a corresponding nested relation:

musicType (name: string, notes: text, band (name: string, photo: picture, album (title: string, cover: picture, song (title: string))))

The structure of a nested relation is represented by a *structure tree* (Massari *et al.*, 1997). A structure tree can be described by recursively composing the two data representation constructs "set_of" and "record". Informally, a "set_of" is an unordered set of elements of the same type; a "record" is a list of elements which can be of different types. One or more elements of a record can be a "set_of". Thus a structure tree is composed of nodes and edges, every node being a "set_of" or a "record" construct. Formally, a structure tree can be recursively defined as follows:

1. D, where D is an atomic domain of values;
2. set-of (T), where T is a structure tree;

3. record A_1:T_1, ..., A_n:T_n end, where the A_i's are distinct symbols, and the T_i's are structure trees.

An example of structure tree concerning the nested relation reported above is given in Figure 2.

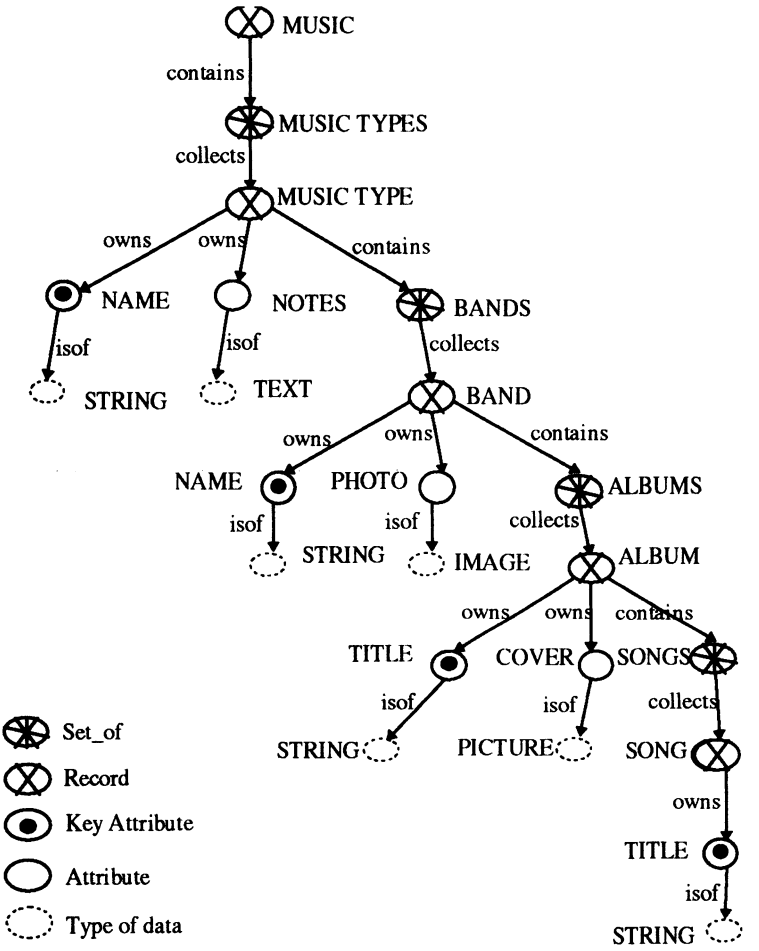

Figure 2. The structure tree for the query 1.

The root node is a record with only one field, whose value is the nested relation MUSIC TYPES resulting from the query. This root node actually indicates the database involved in the query, and might occasionally have some atomic values (called accessories in VIRGILIO) describing the database itself. The relation MUSIC TYPES is a set of records with two atomic values, a string (NAME) and a text (NOTES), and a nested relation (BANDS). The first of the attributes of the relation MUSIC TYPES is considered as a key. Similarly, the

relation BANDS is a set of records with two atomic values, a string (NAME) and an image (PHOTO), and a nested relation (ALBUMS). The interpretation of the rest of the tree is straightforward.

The architecture of the Query Management Tool that is a component of the Virgilio System (see Figure 1) is detailed in Figure 3. Beside the Visual Query Interface the main modules are the Structure Tree Generator and the Prolog Query Generator. The former takes in input an SQL expression, computes the answer set by querying the operational database, transform the answer set into a nested relation by analyzing the structure of the SQL query, and generates the corresponding structure tree. This task is performed by using two tools appropriate for the transformation of structured input, namely Flex and Bison, which are the evolution of the well-known Lex and Yacc, respectively (Levine *et al.*, 1990). Flex is used to implement a scanner of SQL queries, while Bison is applied to build the corresponding parser.

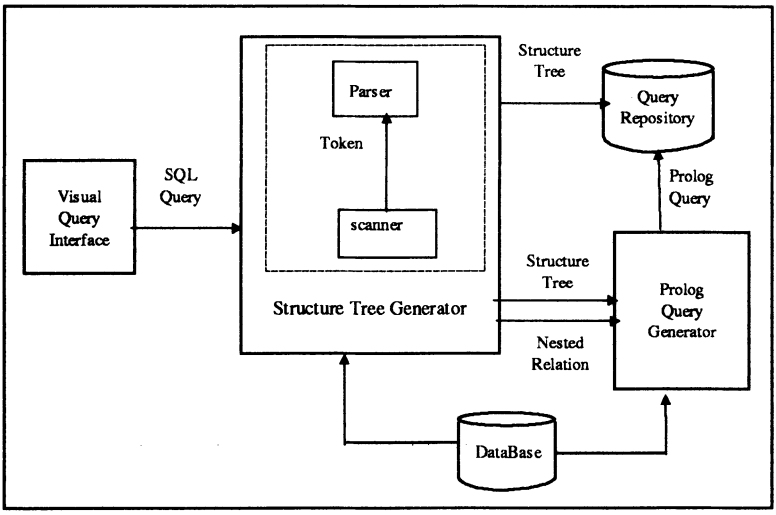

Figure 3. Architecture of the Query Management Tool.

Whenever a sequence of input tokens matches one of the rules in the grammar of SQL queries, an action is taken. Actions concern the selection of relations and attributes involved in the query, the identification of joining attributes, and the nesting based on the clause ORDER BY. The structure tree representing the nested relation is stored in the Query Repository and is passed to the Prolog Query Generator that transforms the structure tree into a Prolog query useful for the mapping process. This module takes in input the nested relation as well, since the query Prolog should include information on the cardinalities of relations composing the nested relation. Finally the Prolog query is stored in the Query Repository.

5.2 Requirements for the mapping

Once the structure tree of a query has been generated, it is necessary to map the structure tree into some virtual world taken from a set of predefined virtual worlds stored into the VR Object Repository. The mapping process has to meet a number of requirements, namely

- *Consistency with the structure-tree*: The metaphor should allow for browsing data according to the hierarchical relations expressed in the structure tree. Indeed, the organization of the results by a structure tree is based on the structure of the SQL query, that is, on the way in which the user has formulated his/her request. When the correspondence between the structure of the SQL query and the "structure" of the virtual world is strict, the user will browse more easily, since he/she already knows the directions to choose.
- *Completeness of the metaphor*: All data in the result relation of a query should be reachable from the starting point of exploration.
- *Realism of the virtual world*: The scenes presented to the user should be fairly realistic. For instance, showing a big wall with hundreds of posters is not the best way to aggregate data concerning lyric singers.
- *Effectiveness of the metaphor:* Properties of objects in the virtual world should match properties of data they represents. For instance, a CD in a virtual scene represents some songs and by clicking on it should be possible to hear a song, while pages of a book are more appropriate to represent the text of songs.

5.3 Categorization of virtual world objects

Whether the result of a query could be represented by a virtual world strongly depends on the variedness of the virtual world. Objects of the virtual world can be categorized into three different classes:

1. *Aggregators*, which do not necessarily represent a piece of data by themselves, but aggregate a set of virtual world objects (virtual objects for short) of different type. For instance, a table can aggregate a book and a picture frame, the former used to represent text of songs while the latter showing the portrait of a singer. A folder is also an aggregator since it can be used to aggregate a variable number of documents of different type.
2. *Classifiers*, which assemble sets of aggregators of the same type. For instance, a chest-of-drawers is a classifier since it contains several drawers. Of course, a classifier may be more appropriate than another for representing a particular set of data. An important factor is the number of aggregators it contains. A real chest-of-drawers contains from two to six drawers, while a book has from eighty to one thousand pages. Therefore, for each classifier it

is necessary to define a minimum and a maximum number of aggregators to be assembled.

3. *Accessories*, which represent a specific type of data. A poster is an example of accessory useful to represent image data, while a label is an example of accessory used to represent a string.

It is interesting to observe that some aggregators may have two different visual representations: *external* and *internal*. The former is called *aggregator symbol*, and is typically shown when the user is browsing a classifier, while the latter is the natural representation of the aggregate, and is shown once the user has chosen an aggregator from a classifier. For instance, a room is an aggregator normally accessed from a corridor (classifier). Its internal representation is obvious and depends from the aggregated data, while its external representation might be a door with a name label on it. Thus, when the user is in the corridor, he/she sees only several doors, but when he/she enters a certain room he can see every object inside.

5.4 Metaphor definition as a constraint satisfaction problem

Our approach toward the automatic definition of a metaphor is based on the fact that knowledge on the virtual worlds can be easily represented in a logical formalism. In particular the universe of discourse concerns virtual objects (aggregators, classifiers, accessories), the aggregator symbols, the types of data, and the integer numbers. Each element of the universe of discourse is identified by a distinct constant. For instance, if we consider the virtual world "building" we can define the following constants:

- AGGREGATORS: *elevator*, room, floor, drawer, folder, page;
- CLASSIFIERS: button-table, chest-of-drawers, album, corridor, folder-collection;
- ACCESSORIES: floor-name, sideboard, poster, board, picture, photo, door-label, drawer-picture, drawer-label, index-item;
- AGGREGATOR SYMBOLS: door, drawer-front, index, button;
- TYPES OF DATA: string, text, picture, image.

Interrelationships between objects of the universe of discourse are expressed by ground facts. Some of the predicates concern the relations reported in a structure tree. They are:

- *contains(*Aggregator, *Classifier)*, stating that Classifier can be contained in Aggregator;
- *collects(Classifier, Aggregator)*, stating that Classifier can collect a set of Aggregators;
- *owns(Aggregator, Accessory)* or *owns(AggregatorSymbol, Accessory)*, stating that Aggregator (Symbol) can contain an Accessory;

- *is-of(Accessory, TypeOfData),* defining the type of data associable to Accessory.

Moreover, the predicate *hasicon(Aggregator, AggregatorSymbol)* defines the Aggregator Symbol associated to an Aggregator, while predicates *hasmin(Classifier, Integer)* and *hasmax(Classifier, Integer)* are used to express realistic constraints on the number of objects aggregated by a classifier. This prevents the generation of fictitious scenes, e.g. buildings with thousands of floors or books with one page. If it is necessary to represent hundreds of tuples, a more appropriate virtual world will be selected by the system.

An example of possible interrelationships between objects of the virtual world "building" is reported in Figure 4.

Given a background knowledge on some virtual worlds, the problem of mapping the structure tree into some virtual world can be cast as a problem of constraint satisfaction. As shown by Mackworth (1977), one way to view a constraint satisfaction problem is as the problem of providing a constructive proof of the validity of a formula (without function symbols) of the form F→G, where F is the world description expressed as a conjunction of ground literals listing all facts in the world, while G is the goal, that is an existential formula with a conjunction of literals (the constraints). The constructive proof of formula's validity automatically yields a substitution for the existential variables.

```
contains(elevator,
        button-table).
contains(room,
        chest-of-drawers).
contains(room, album).
contains(floor, corridor).
contains(drawer,
        folder-collection).

collects(button-table,floor).
collects(corridor,room).
collects(chest-of-
drawers,drawer).
collects(folder-collection,
        folder).
collects(album,page).

hasmin(button-table,2).
hasmin(corridor,1).
hasmin(chest-of-
drawers,0).
hasmin(folders,1).
hasmin(album,1).
hasmax(button-table,20).
hasmax(corridor,30).
hasmax(chest-of-drawers,
        20).
hasmax(folder-collection,
        10).
hasmax(album,30).

owns(floor,floor-name).
owns(floor,sideboard).
owns(room, poster).
owns(folder,photo).
owns(room,door-label).
owns(drawer,drawer-label).
owns(drawer,
        drawer-picture).
owns(door,door-label).
owns(drawer-front,
        drawer-picture).
owns(drawer-front,
        drawer-label).
isof(floor-name,string).
isof(drawer-label,string).
isof(door-label,string).
isof(button-label,string).
isof(sideboard,text).
isof(board,text).
isof(picture,picture).
isof(drawer-
picture,picture).
isof(photo,picture).
isof(poster,image).

hasicon(room, door).
hasicon(drawer,
        drawer-front).
hasicon(page, index).
hasicon(floor, button).
hasicon(elevator,nil).
hasicon(folder,nil)
```

Figure 4. An example of background knowledge on the virtual world "building".

In our specific application, F is the description of a virtual world (e.g., "building"), while G is the description of the structure tree, which is represented as a conjunction of nonground literals whose variables are existentially quantified. Such literals describe the structure of the query, and define the actual number of elements aggregated by a classifier. The same predicate symbols above are used to describe a structure tree. For instance, the logical formulation of the structure tree in Figure 2 is reported in Figure 5, where the integer numbers are cardinalities of the relations composing the nested relations determined by the query.

From a practical point of view, we can use Prolog interpreters to prove formula's validity (Shalkoff, 1990). The logic program is the set of ground facts of the background knowledge, while the Prolog query is the logical formulation of the structure tree. The answer computed by the Prolog interpreter defines the instantiations of the variables in the Prolog query with some virtual objects. In this way the mapping of the structure tree into some virtual world is completely defined, that is the metaphor is eventually generated. As an example, the answer computed for the query in Figure 5 will define the following instantiations reported in Figure 6.

∃ contains(RecordMusic, SetofMusicTypes) ∧ hasicon(RecordMusic,RecordMusicIcon)∧
collects(SetOfMusicTypes, RecordMusicType) ∧
hasicon(RecordMusicType, RecordMusicTypeIcon) ∧
hasmin(SetOfMusicTypes, MinMusicTypes) ∧ 7 >= MinMusicTypes ∧
hasmax(SetOfMusicTypes, MaxMusicTypes) ∧ MaxMusicTypes >= 7 ∧
owns(RecordMusicType, AttributeNameType) ∧ isof(AttributeNameType, string) ∧
owns(RecordMusicType, AttributeNotes) ∧ isof(AttributeNotes, text) ∧
contains(RecordMusicType, SetOfBands) ∧ collects(SetOfBands, RecordBand) ∧
hasicon(RecordBand, RecordBandIcon) ∧ hasmin(SetOfBands, MinBands) ∧
2 >= MinBands ∧ hasmax(SetOfBands, MaxBands) ∧ MaxBands >= 30 ∧
owns(RecordBand, AttributeName) ∧ isof(AttributeName, string) ∧
owns(RecordBand, AttributePhoto) ∧ isof(AttributePhoto, image) ∧
contains(RecordBand, SetOfAlbums) ∧ collects(SetOfAlbums, RecordAlbum) ∧
hasicon(RecordAlbum, RecordAlbumIcon) ∧ hasmin(SetOfAlbums, MinAlbums) ∧
1 >= MinAlbums ∧ hasmax(SetOfAlbums, MaxAlbums) ∧ MaxAlbums >= 8 ∧
owns(RecordAlbum, AttributeTitle) ∧ isof(AttributeTitle, string) ∧
owns(RecordAlbum, AttributeCover) ∧ isof(AttributeCover, picture) ∧
contains(RecordAlbum, SetOfSongs) ∧ collects(SetOfSongs, RecordSong) ∧
hasicon(RecordSong, RecordSongIcon) ∧ hasmin(SetOfSongs, MinSong) ∧
1>=MinSong ∧ hasmax(SetOfSongs, MaxSong) ∧ MaxSong >=10 ∧
owns(RecordSong, AttributeNameSong) ∧ isof(AttributeNameSong, string)

Figure 5. Logical formulation of the structure tree in Figure 2.

RecordMusic ← elevator
RecordMusicIcon←nil
SetOfMusicTypes←button-table
MinMusicTypes←2
MaxMusicTypes←20
RecordMusicType←floor
RecordMusicTypeIcon←button
AttributeNameType←floor-name
AttributeNotes←sideboard
SetOfBands←corridor
MinBands←1
MaxBands←30
RecordBand←room
RecordBandIcon←door
AttributeName←door-label
AttributePhoto←poster
SetOfAlbums←chest-of-drawers
MinAlbums←0
MaxAlbums←20
RecordAlbum←drawer
RecordAlbumIcon←drawer-front
AttributeTitle←drawer-label
AttributeCover←drawer-picture
SetOfSongs←folder-collection
MinSongs←1
MaxSongs←10
RecordSong←folder
RecordSongIcon←nil
AttributeNameSong←folder-name

Figure 6. Instatiations of the variables of the query in Figure 5 that define the mapping between structure tree and virtual world.

It is interesting to observe that there might be several answers for the same Prolog query, that is several metaphors to represent the results of the same SQL query. When this happens, it is important to have a criterion for choosing one of the possible metaphors. The choice might be based on the most recent virtual world used to answer a query of the same user. The underlying assumption is that the user is more familiar with the most recently visualized virtual world. On the contrary, we have defined no criterion for choosing among metaphors concerning the same virtual world, since any reasonable choice should be based on a user model not currently available in Virgilio.

5.5 The metaphor definition tool

In Virgilio, the mapping process described above is performed by the Metaphor Definition Tool (MDT). The MDT takes in input both the background knowledge on the virtual worlds and the Prolog query, and produces a *metaphor graph* that associates each node of the structure tree with some virtual object. The Virtual World Object Repository is the database where background knowledge on the virtual worlds is stored, while the Query Repository stores the structure trees and their corresponding Prolog queries. The metaphor graph is stored in the Metaphor Repository, so that it can be retrieved by the Scene Constructor Server that builds the sequence of scenes of the chosen virtual world that visualizes the query results. The scenes are constructed by using VRML, but the work performed by the Scene Constructor Server is out of the scope of this paper.

Two examples of scenes built using a prototype of Virgilio are depicted in Figures 7 and 8 (Paradiso, 1997). They concern query 1 whose result has been

mapped into the metaphor "building". The user may browse the query result by walking into the building from its entrance to the different floors and rooms.

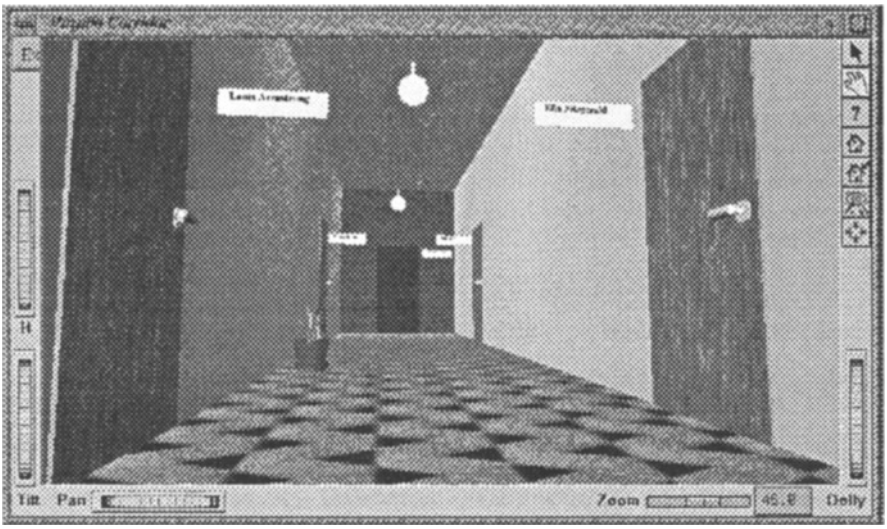

Figure 7. One of the scenes visualizing the result of query 1. The scene shows a floor of the "building".

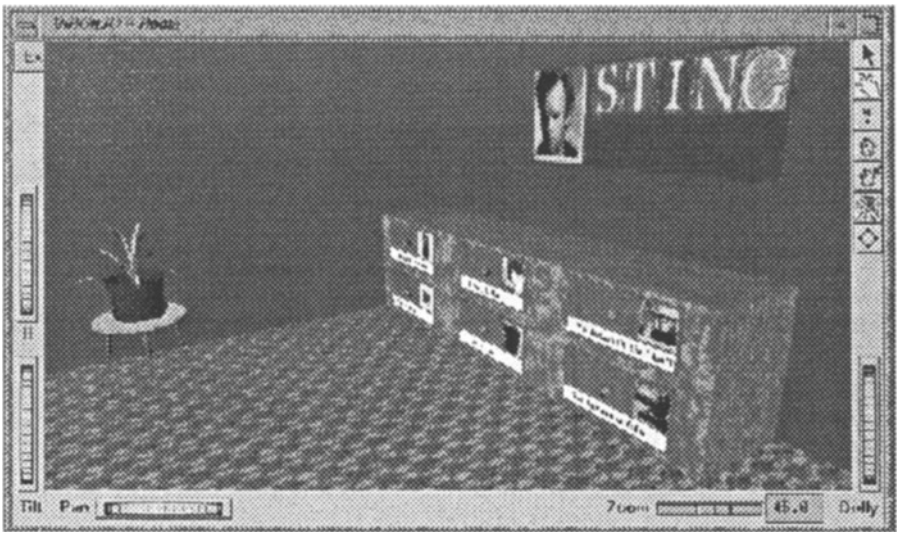

Figure 8. Another scene visualizing the result of query 1. It shows the inside of one of the rooms whose doors are visible in the scene in Figure 7.

The scene in Figure 7 shows a floor of the "building"; the floor represents a type of music. The corridor provides access to different rooms, each one associated to a band. The names of the bands are written on the door labels. In Figure 8 we see another scene shown the inside of one of the rooms whose doors are visible in the scene in Figure 7. The users can see different objects representing he information related to the band. In this examples the band is actually the famous singer "Sting" whose picture is shown in a poster on the wall, together with his name. The drawers contain albums of such a singer.

6 CONCLUSIONS

In this paper we have presented a novel approach to automatically generate metaphors to be exploited in the interaction between users and databases. The metaphor definition problem has been treated as a constraint satisfaction problem, that is viewed as the problem of providing a constructive proof of the validity of a formula.

The work discussed in this paper is a further step in the development of Virgilio, a Virtual Reality (VR) based system that visualize objects in a database through effective VR techniques. We have illustrated the process that generates from a query to the database all information necessary to the construction of the VR scene that visualizes the query results, also allowing the users to conveniently browse such results.

As future work, we are planning to incorporate a user model into the system, so that it can provide further knowledge to be exploited in the choice of the metaphor. Moreover we are planning some more accurate testing of the current prototype with end users, from which we can should get significant feedback in our design.

ACKNOWLEDGMENTS

The authors appreciate Annabella Loconsole and Marcello L'Abbate for their helpful collaboration on the implementation of parts of Virgilio during their stage at the Department of Computer Science of the University of Bari and at in the GMD-IPSI Institute of Darmstadt.

REFERENCES

Atzeni, P. and de Antonellis, V. (1993) Relational Database Theory. Benjamin/ Cummings, Redwood City, CA.

Catarci, T., Costabile, M.F. and Matera, M. (1995) Visual Metaphors for interacting with Databases. *ACM SIGCHI Bulletin*, **27**(2), 15-17.

Catarci, T., Costabile, M.F., Levialdi, S. and Batini, C. (1997) Visual Query Systems for Databases: A Survey. *Journal of Visual Languages and Computing*, **8**, 215-260.

Chang, S.K. and Costabile, M.F. (1997) Visual Interface to Multimedia Databases, in *The Handbook of Multimedia Information Management* (eds. W.I. Grosky, R. Jain, and R. Mehrotra), Prentice Hall, Upper Saddle River, NJ, 167-187.

Erickson, T.D. (1990) Working with Interface Metaphors, in *The Art of Human-Computer Interface Design* (ed. B. Laurel), Addison Wesley, Reading, MA, 65-73.

Gershon, N., Card, S. and Eich, S.G. (1997) Information Visualization, in *Chi 97 Tutorial Notes*, Atlanta, USA, 22-27 March 1997.

Haber, E. M., Ioannidis, Y. E. and Livny, M. (1994) Foundation of Visual Metaphors for Schema Display, *Journal of Intelligent Information Systems*, **3**, 1-38.

Lakoff, G. and Johnson, M. (1980) Metaphors We Live By. The University of Chicago Press, Chicago.

Levine, J., Mason, T. and Brown, D. (1992) Lex & Yacc, 2nd edition. O'Reilly and Associates, Sebastopol, CA.

Mackworth, A.K. (1977) Consistency in networks of relations, *Artificial Intelligence*, **8**, 99-118.

Madsen, K. H. (1994) A Guide to Metaphorical Design, *Communications of the ACM*, **37**, 57-62.

Marcus, A. (1994) Managing Metaphors for Advanced User Interface, *Proceedings of International Workshop AVI'94*, ACM Press, New York, 12-18.

Martin, J.H. (1990) A Computational Model of Metaphor Interpretation. Academic Press, San Diego.

Massari, A., Saladini, L., Hemmje, M. and Sisinni, F. (1997) Virgilio: A Non-Immersive VR System To Browse Multimedia Databases, *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, IEEE Computer Society Press, Los Alamitos, CA, 573-580.

Mountford, S.J. (1990) Tools and Techniques for Creative Design, *in The Art of Human-Computer Interface Design* (ed. B. Laurel), Addison Wesley, Reading, MA, 17-30.

Paradiso, A. and Hemmje, M. (1997) A Generic Refinement of the Virgilio System's Design and a Prototypical Architecture. GMD Technical Report, Nr. 1093, September 1997.

Robertson, G.G., Card, S. K. and Mackinlay, J.D. (1993) Nonimmersive Virtual Reality, *IEEE Computer*, **26**(2), 81-83.

Scholkoff, R.J. (1990) Artificial Intelligence: An engineering Approach, Mc Graw Hill, New York.

BIOGRAPHY

Maria F. Costabile received the Lurea degree in Mathematics at the Universita' della Calabria. Since 1989 she is associate professor at the Department of Computer Science of the University of Bari, Italy. From 1978 to 1988 she worked at the Dipartimento di Matematica, Universita' della Calabria. She has been visiting scientist in several foreign universities. Her current interests include theory of visual languages, visual interfaces, visual languages for querying databases, human-computer interaction, usability of interactive systems, user models. She has published several papers on the above topics, and edited four books. She served in committees of international conferences, and as program co-chair of AVI'96 and AVI'98. She is member of ACM, IEEE, AICA, IAPR. She is chairing the Italian Chapter of ACM SIGCHI.

Matthias Hemmje is a member of the OASYS information systems research division at GMD-IPSI in Darmstadt, Germany. He is working in various projects related to object-relational Multimedia Database Management Systems. Currently, he is responsible for R&D of the ICE DataBlade Module (a database supported Information Catalogue Environment enabling navigation on multimedia documents), as well as a VRML-, an MPEG-, and an SGML-DataBlade Module. He has been conducting the design and development and evaluation of LyberWorld and VIRGILIO prototypes, both 3D graphical information visualization systems. Besides Multimedia Information Systems, his research interests include computer human interaction and information visualization for information systems.

Donato Malerba received the Laurea degree in Computer Science from the University of Bari, Italy, in July 1987. In 1991 he joined the University of Bari where he currently holds the rank of Assistant Professor in the Department of Informatics. In 1992, he has been a visiting scientist at the Department of Information and Computer Science, University of California at Irvine. His research interests are in machine learning, artificial intelligence and pattern recognition. Applications include document classification and understanding, knowledge discovery in databases, map interpretation and interfaces. He has published more than forty papers in international journals and refereed conference proceedings.

Aldo Paradiso received the Laurea degree in Computer Science from the University of Bari, Italy. He has been employed at the GMD-IPSI Institute in Darmstadt, Germany, where he is currently working on his PhD thesis. During his stay at GMD he has worked on the Virgilio Project, also creating a prototype of the system. His research interests include information visualization, particularly using Virtual Reality approaches. At the present he is working on Human Modeling using VR approaches.