GMD Report 15

GMD –
Forschungszentrum
Informationstechnik
GmbH

Marcello L'Abbate, Matthias Hemmje

VIRGILIO
The Metaphor Definition Tool

May 1998

GMD –
Forschungszentrum Informationstechnik GmbH
Schloß Birlinghoven
D-53754 Sankt Augustin
Germany
Telefon +49 -2241 -14 -0
Telefax +49 -2241 -14 -2618
http://www.gmd.de

Anschrift der Verfasser/Address of the authors:
Marcello L'Abbate
Matthias Hemmje
Institut für Integrierte Publikations- und Informationssysteme
GMD – Forschungszentrum Informationstechnik GmbH
Dolivostraße 15
D-64293 Darmstadt
E-mail: labbate@gmd.de
hemmje@gmd.de

ISSN 1435-2702

ABSTRACT: The *Virgilio* software system is a general purpose exploration tool for large collections of complex structured data. It generates visualisations of the results of a query to a database by the help of virtual reality objects, embedded into predefined virtual reality scenarios. This document describes the design, specification, and implementation of the Metaphor Definition Tool. This component of the *Virgilio* software system has to identify the most suitable virtual worlds for representing the result of a query. This is achieved by considering the metaphor definition task as a general constraint satisfaction problem. The logical formalism adopted to describe interrelationships among the involved objects corresponds to that of the Prolog programming language, in order to take advantage from its built in backtracking strategy. This paper presents the overall *Virgilio* architecture, provides a brief introduction to the concept of metaphor and its use within the *Virgilio* environment, contains a description of the MDT as well as a description of the repositories, and explains how to integrate the MDT into the whole system. Two complex mapping examples are also introduced.

Keywords: Information systems, user interfaces, information visualisation, metaphor generation

INDEX

1. INTRODUCTION

Virgilio is a VR-based system, which generates visualisations of complex data objects representing the result of a query [Massari et al. 96, Massari et al. 97, Massari et al. 98]. The system uses a data base repository of virtual world objects and creates a visual representation of the data-set resulting from the formulated query. This is achieved by following precise rules. The structure of the query result must be preserved and the chosen virtual objects must be suitable for supporting the type of data to be displayed. Moreover, *Virgilio* is based on concrete metaphors (i.e. metaphors where the visual domain is composed by objects known by users from their everyday experience) in order to take advantage of common knowledge about real world objects.

Two different classes of users may interact with the *Virgilio* system: final users and system administrators. Final users, on the one hand, will access the system by formulating a query by means of natural language. After the initial information retrieval step of the *Virgilio* system, they will browse the data items by means of a virtual reality exploration. System administrators, on the other hand, have to carry out mainly two tasks. First they will translate the query formulated by the final users into an SQL-form, in order to allow further processing within the system. Second, a system administrator takes also part in the definition of the actual set of virtual scenes to be presented to the final users. It is possible to think of automatic procedures that may support or even substitute this work of system administrators. This issue is discussed later in this document.

Another relevant feature of the *Virgilio* system is the possibility to use it like a standard client-server application on the web. The virtual world scenes are stored in VRML format [VRML], and the final users can access the system by means of a common web-browser which supports VRML.

A component of the *Virgilio* system, called the Metaphor Definition Tool (MDT), generates and analyses all possible mappings among the objects of the query data-set and the objects of a set of virtual worlds. Further procedures will determine whether the generated mappings are to be considered as efficient metaphors or not, and therefore to be shown to the user as the final query result visualisation of *Virgilio*.

This document describes the MDT design, specification and its implementation. Section 2 presents the overall *Virgilio* architecture. Section 3 provides a brief introduction to the concept of metaphor and its use within the *Virgilio* environment. Section 4 contains a description of the MDT as well as a description of the repositories. The mapping process is discussed in the 5th section. Finally section 6 explains how to integrate the MDT into the whole system. The appendix introduces two complex mapping examples.

2. OVERVIEW OF THE ARCHITECTURE OF *VIRGILIO*

In Figure 2.1 the architecture of *Virgilio* is depicted. The final user's query is first processed by the Query Tool (QT). It provides an SQL script of the query which is stored into the Query Repository (QR) as output. The MDT will take the selected query as input, as well as data taken from the Virtual World Object Repository (VWOR) in which the virtual objects are described, in order to find out one or more metaphors suitable for representing the result of the query into a virtual scene. The MDT will produce data to be stored into the Metaphor Repository (MR). This data will be used by the Scene Constructor (SC) to actually build the VRML scenes. The SC will also access data stored in other repositories, that means:
- the VRML frameworks, from the VWOR;
- the selected query script from the QR;
- the actual result data-set from the source database.

The built scenes will finally be presented to the user who will navigate through the resulting visualisation by using a common web browser.

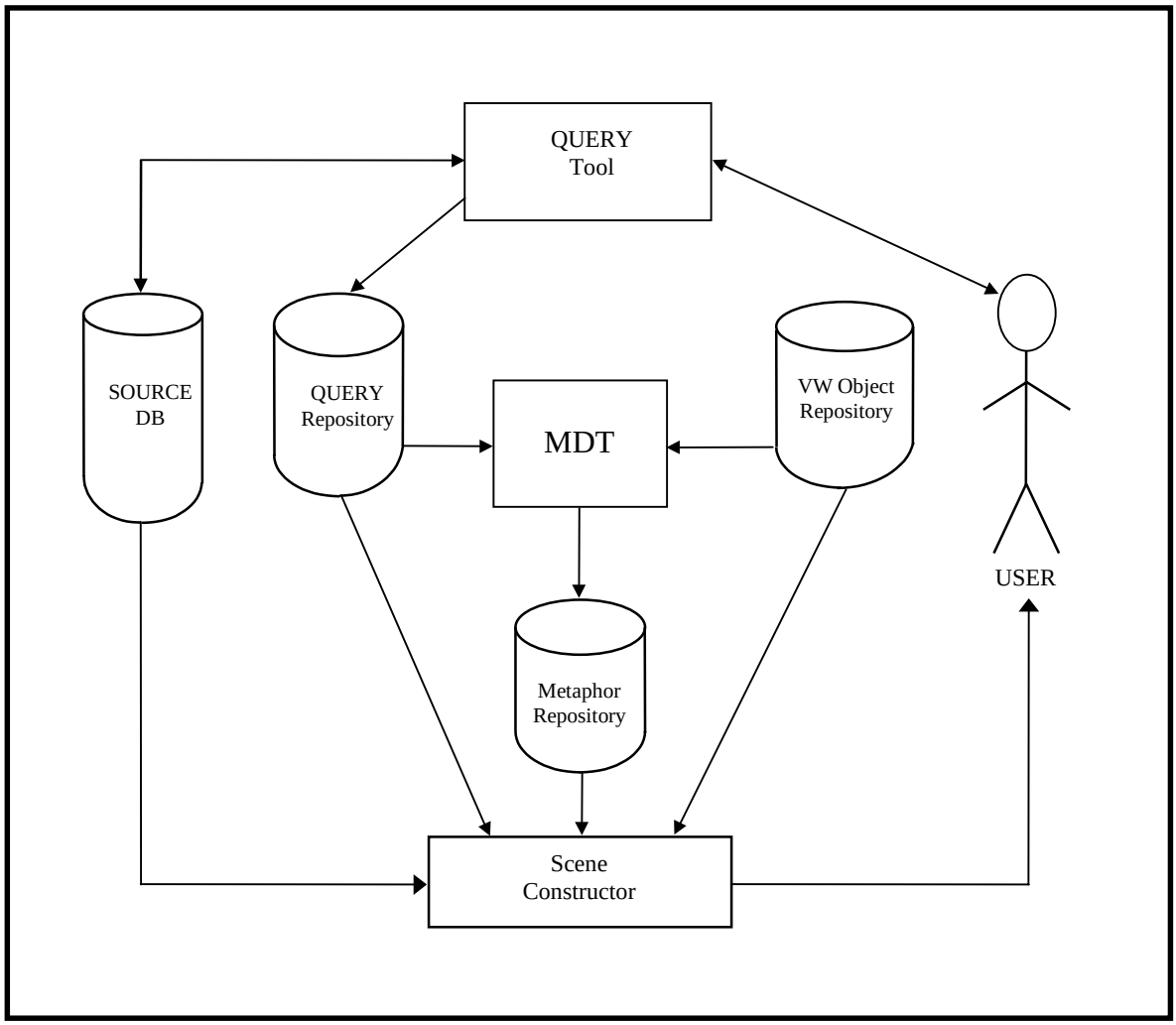


Figure 2.1 - Architecture of *Virgilio*

Please note that the presence of a system administrator has been omitted in this overview of the architecture for two reasons. First, the interpretation and understanding of the system becomes easier and clearer; second, as already mentioned, it is possible to design procedures that may substitute the system administrator's work, and the schema of figure 2.1 would be a possible solution for that topic. The QT can be realised like a standard visual query interface [Catarci and Costabile 95 , Massari et al. 95] producing an SQL script for the selected query. Otherwise naive final users will not be able to formulate their request in natural language. Moreover, as we will see in the next section, the definition of the actual set of virtual scenes to be presented to the user, can be entirely carried out by the MDT.

Typically, the data composing the query result has a structure as complex as the modelling primitives allowed by the database model. *Virgilio* operates with those models which can support, directly or indirectly, the notion of nested relations [Atzeni and De Antonellis 93]. Informally, a nested relation is a set of tuples such that the values of attributes are allowed to be nested relation themselves.

The queries stored in the query repository (i.e. their SQL expression) will also be represented, in the same repository, in the form of a *Structure Tree* [Massari et al. 96]. A structure tree is a directed graph, whose edges represent the nested relations of the query. The structured-tree queries will be used either by the MDT or by the Scene Constructor for their purposes. The structure tree appearing in this paper has a slightly different implementation than the one described in [Massari et al. 96]. Section 4.2 will describe these differences, as well as the structure tree's use within the system.

3. WHY METAPHORS?

Metaphors are very much used in natural language. A definition given by Lakoff and Johnson states that "a metaphor is a rhetoric figure, whose essence is understanding and experiencing one kind of things in terms of another" [Lakoff and Johnson 80]. Therefore, a metaphor gives the possibility to understand new and complex concepts by means of more familiar (i.e. well-known) ones. This feature has been exploited in the interfaces of several computer systems. Indeed, in the last two decades the tendency of the computing world has been to concentrate into designing and developing powerful interfaces between computer and end-user [Carroll et al. 88, Marcus 94, Erickson 90]. All of them strictly rely on the metaphor concept, thus even the inexperienced user may think of common actions while performing complex and delicate tasks on a machine.

It is possible to identify two sets of component concepts in every metaphor, namely target and source concepts [Martin 90]. The former consists of the concepts to which the metaphor actually refers to, the latter consists of the concepts used in substitution of the target concepts. For instance, if we consider the desktop interface-metaphor, used in several kinds of computer platforms, we can notice an often-used wastebasket-tool, in which the action of throwing away some paper (source concept) represents a familiar way for deleting an item from the memory or hard disk (target concept).
The metaphors used by *Virgilio* go in a slightly different direction, e.g. in a data-oriented way. Indeed, the target concepts consist not only of actions such as the interface metaphors, but they also include final information, namely different kinds of data. Many authors have already done some research on that topic [Haber et al. 94, Costabile et al. 95, Aloia et al. 96], and the *Virgilio* system provides a combination between action and data-oriented metaphors, generated in a semi-automatic way.

But how can one evaluate if the metaphors created by *Virgilio* are useful and efficient? It is very difficult and arduous to give a precise answer to this question. Many authors and computer designers proved the efficiency of a metaphor by means of usability tests [Marcus 94, Carroll et al. 88]. They submitted a metaphor to a selected group of different users, studying and examining the results in a statistic way. But this is hardly possible with the metaphors used by *Virgilio*. Even if the source concepts (e.g. the available virtual worlds) are numerous but fixed as regards the number of occurrences, the target concepts may vary in every user-session, producing an infinite number of different metaphors to be tested.

An alternative solution for evaluating metaphors was given by Gentner [Gentner 80] and then discussed by Carroll, Mack and Kellog [Carroll et al. 88]. He has defined a set of characteristics such as base specificity, clarity, richness and so on, which have to be satisfied by every metaphor.

These characteristics are:

„BASE SPECIFICITY“.....is defined as the extent to which the structure of the metaphor source domain is explicitly understood (i.e., in terms of its predicates). It sets an upper limit on the potential usefulness of the source domain as a predictor of the target.

„CLARITY“.....refers to the precision of the node correspondences across the mapping; a metaphor where node and predicate correspondences are one-to-one rather than, for example, one-to-many in the target domain is higher in clarity.

„RICHNESS“.....is the density of predicates (relative to the total specified in the base) included in the mapping.

„ABSTRACTNESS“.....refers to the level at which relations comprising the mapping are defined. If they are the first order predicates (relation among nodes) of the metaphor source, the mapping is less abstract than if they are higher order relations (those among predicates) in the source domain.

„SYSTEMATICITY“.....metaphors are systematic to the extent that mapped relations are mutually constrained by membership in some structure of relations.

„EXHAUSTIVENESS“.....base exhaustive metaphors map each of their relations into the target domain (target exhaustive metaphors are defined analogously).

„TRANSPARENCY“.....metaphors are more transparent if it is easy to determine which of the relations specified in the base are to be mapped into the target.

„SCOPE“.....refers to the extensibility of the mapping.

It is possible to think of automatic procedures that may evaluate most of the constraints proposed by Gentner.
Another element to take into consideration while generating a metaphor surely is the user model [Chang et al. 93]. A metaphor is more efficient if it fits the user's needs and habits well. In future versions of *Virgilio* users will introduce themselves to the system by answering a given form, about their attitudes and working areas, in order to create adequate metaphors. The system will be expanded with a new repository (the user repository, UR), which will contain all models of users that have previously interacted with *Virgilio*.

4. THE MDT

The MDT is a module of the *Virgilio* system, which has to identify, among all predefined virtual worlds, the most suitable ones for representing the query result. A virtual world is defined to be suitable when it meets a set of requirements. For instance, a virtual world has to contain objects capable of displaying all data types of a potential query result. Moreover, as already mentioned, it is necessary to preserve the structure of the formulated query which has been imagined and therefore determined by the final user himself. Navigation through the resulting information visualisation will seem more natural to him, since he already knows the directions to choose and how “deep” to go on while browsing the visualised data. Finally, a virtual world has to guarantee the above described metaphoric effect, in order to allow a more spontaneous and effective browsing of the data.

4.1 Mappings

The mapping process of the MDT will determine whether a virtual world is suitable or not for visualising the current query. This is achieved by examining the objects composing the virtual world in order to generate one-to-one correspondences with all elements composing the query result data-set. Such a correspondence will be established only if the involved objects support the same kind of data and only if they maintain the same position within the overall structure schema.

For example, consider a simple virtual world composed by a room with a poster and a chest of drawers inside. The poster may visualise a large image and the chest of drawers may be composed of a variable number of drawers. Each drawer may contain folders displaying some text. Moreover, consider a query retrieving information about a music singer, let's say Sting. This information would comprehend his photograph and all his released Compact Disks. The query would also retrieve the titles of the songs contained in each CD. If we now apply the mapping process the following result may be determined:

- the whole room may represent the singer Sting;
- the poster on the one wall would visualise his photograph;
- the chest of drawers may contain a number of drawers corresponding to the number of CD's retrieved;
- each drawer would be mapped to a specific CD;
- by opening one drawer you would find some folders, each of them corresponding to a song contained into the related CD.

The mapping process is successful in this case because the two structures correspond to each other. Moreover the objects of the room displaying final information (e.g. the poster and the folders in a drawer) support a kind of data (e.g. an image and some text) compatible with the kind of data of the retrieved information (e.g. the singer's photo and the titles of songs).

This virtual world may be expanded by considering a corridor of rooms. Each room may contain the same objects described above. Therefore, it would be possible to formulate a query retrieving a set of singers. Each of them would be mapped to a single room of the corridor.

In figure 4.1 a scheme of the MDT is sketched. The following modules can be seen in the figure:

- *MDT core*, which performs the mapping process;
- *Query repository interface*, which transforms the structure tree of a query into a suitable format for the MDT (e.g. a list, see section 5.2);
- *VW object interface*, which provides the MTD Core with the necessary data describing the virtual worlds (e.g. their overall structure and the kind of data supported by each single object);
- *Metaphor repository interface*, which stores the result mappings produced by the MDT Core into the MR.

The List Generator provides an internal description (i.e. with respect to the MDT) of the containment relationships of the VW objects. This description is also a list (see below, in section 5).

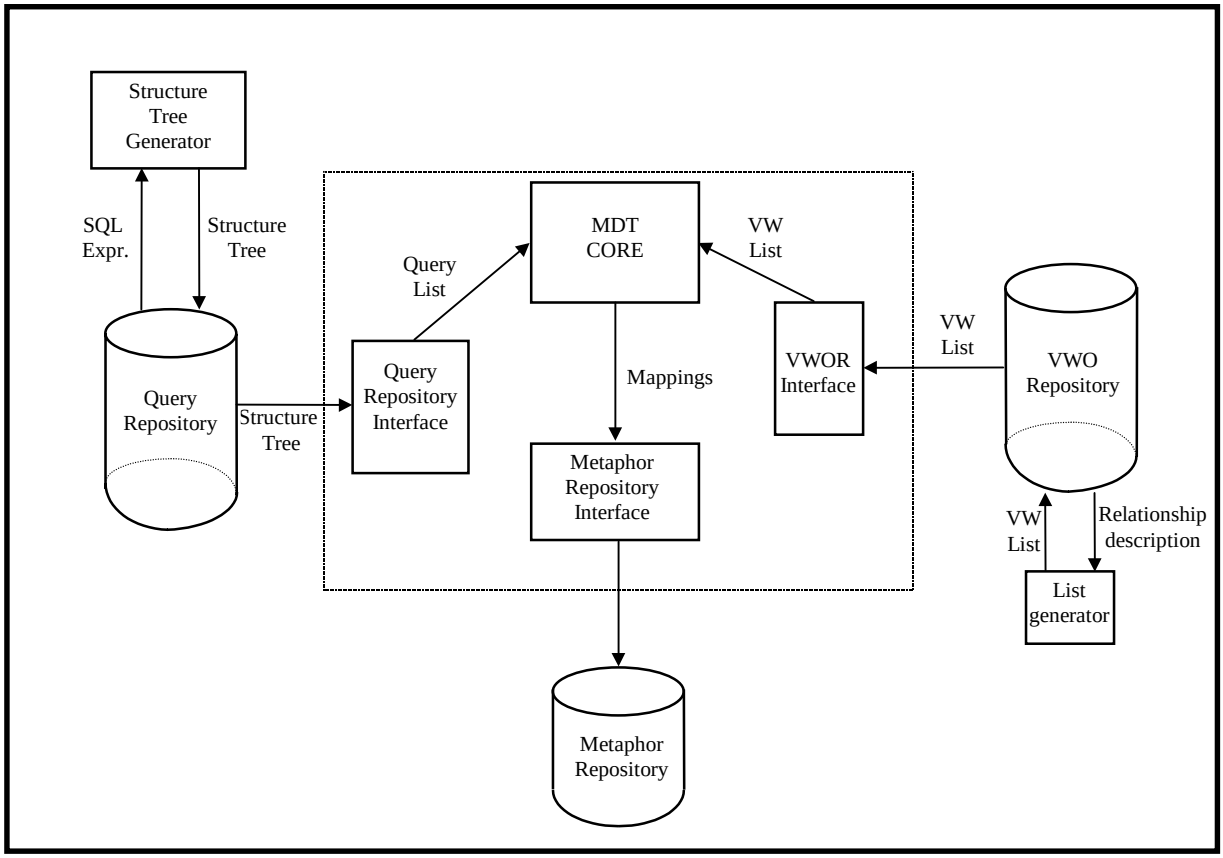


Figure 4.1 - The components of the Metaphor Definition Tool

4.2 Query Repository

Whatever interface is provided for formulating a query (i.e. by means of the work of a system administrator or using a visual query interface), such a query is translated into an SQL form. Another component of the system, called Structure Tree Generator (STG, see figure 4.1), divides the query into sub-queries and saves them in form of a structure tree in the Query Repository (QR). Figure 4.2 shows the ER-schema of the QR. Every structure tree Node is instanced to one of the following three node classes:
- a *set_of* node, which contains a sub-query retrieving a finite set of identical objects;
- a *record* node, which contains a sub-query retrieving an aggregation of different objects;
- an *attribute* node, which contains a sub-query retrieving a specific data item.

A complex query can thus be described by a combination of nodes belonging to the above mentioned classes. They are linked together by containment relations in order to generate different levels of nesting.
Every query starts with a record node (see the "*root*" relation in Figure 4.2) that may be tied either to set_of nodes (by means of the "*contains*" relation), or to attribute nodes (by means of the "*owns*" relation), or at last to other record nodes (by means of the "*has*" relation). The objects retrieved by the sub-query of a set_of node are of kind record node (see the "*collects*" relation in Figure 4.2). Since attribute nodes may be of different data types, depending on the kind of information they retrieve (i.e. text, picture, sound...), they are linked by the relation "*isof*" to another entity of the QR, called *typeofData*.
Each query is identified by the attribute “Qname”.

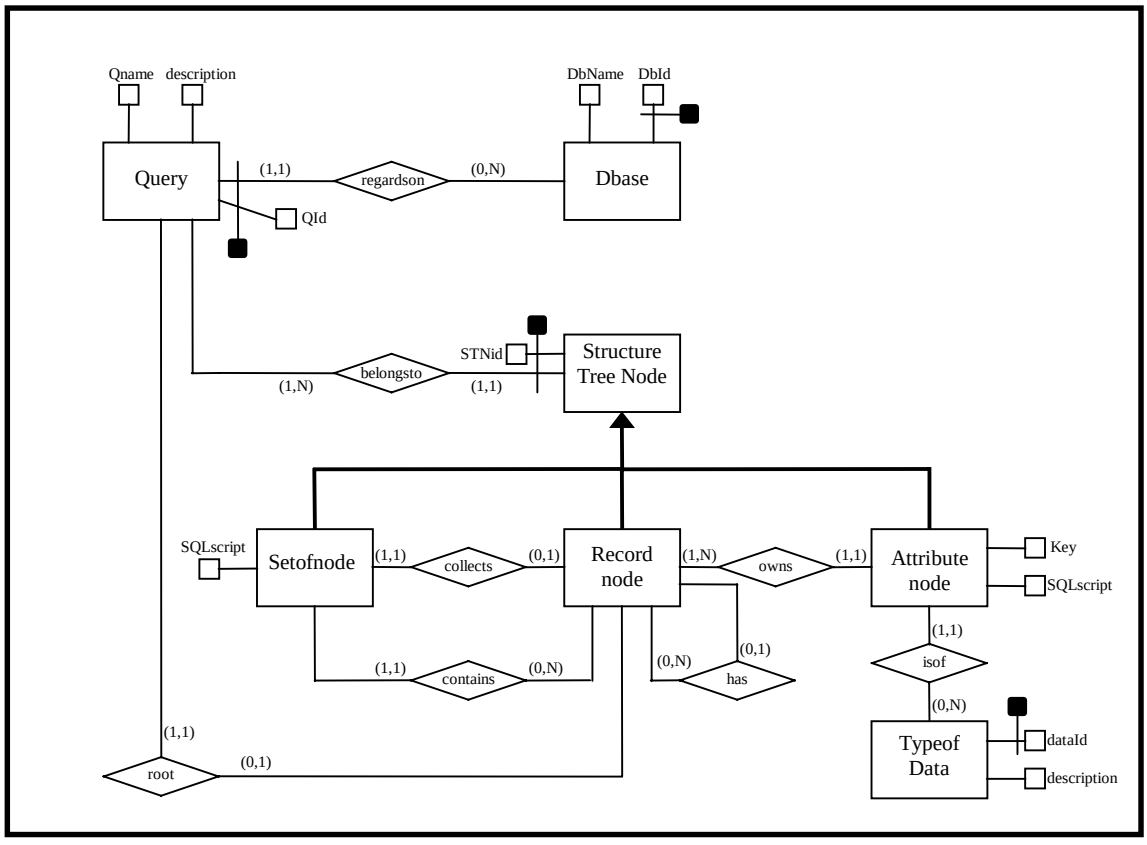
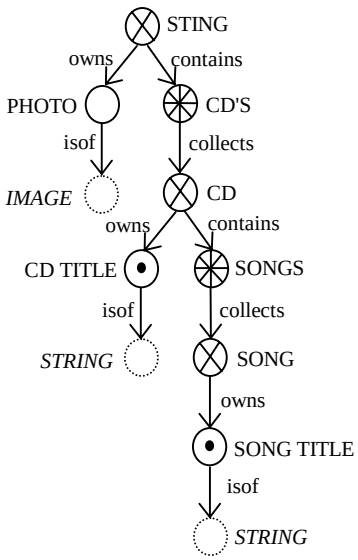


Figure 4.2 - Query Repository

The query of the example in section 4.1 would have a structure tree corresponding to the one contained in figure 4.3.



SYMBOL	STRUCTURE TREE NODE
⊛	Set_of node
⊗	Record node
⊙	Key Attribute node
○	Attribute node
◌	Type_of_data node

Figure 4.3 - Structure tree query "Sting"

The structure tree proposed in this paper has a slightly different implementation than the one specified in [Massari et al. 96]. In this new version a naming of the edges has been introduced, with respect to the following schema:

- if an instance of a record node has got an instance of a set_of node as a child, they will be linked together by an edge named "contains";
- if an instance of a record node has got an instance of another record node as a child, they will be linked together by an edge named "has";
- if an instance of a record node has got an instance of an attribute node as a child, they will be linked together by an edge named "owns";
- if an instance of a set_of node has got an instance of a record node as a child, they will be linked together by an edge named "collects".

The names chosen for the edges correspond to the names of the relations contained in the ER-schema of the QR (figure 4.2).

Moreover, in this newly proposed version of a structure tree, "type_of_data" nodes have also been inserted, in order to have a direct access to the kind of data that an attribute node may support (i.e., text, image, sound, and so on). This information will take part in the final determination process of the virtual scenes to be presented to the user. It is therefore necessary to define a standard of possible data types supported by the whole system.

It is now possible to directly transform the structure tree in a suitable format for the MDT core component. This operation will be carried out by the query repository interface (see figure 4.1 and section 5 for a detailed explanation).

4.3 Virtual World Object Repository

The virtual scenes which will visualise the output of the *Virgilio* system are implemented in [VRML] in order to allow a standard client-server application on the Web. The scenes are stored into the Virtual World Object Repository (VWOR), whose ER-schema is depicted in Figure 4.4.
The objects composing a complex virtual world may be of one of the following kinds:
- *Aggregator*, which represents a scene capable of containing different other VR objects;
- *Classifier*, which represents a scene displaying a set of identical Aggregator objects;
- *Accessory*, which represents an object supporting a specific kind of data;
- *Aggrsymbol*, which can be used within a VR scene as an icon for an Aggregator.

Aggrsymbols are often used for switching from one scene to another.
An aggregator object may contain some classifiers (linked with the relation "contains"), some accessory objects (linked with the relation "owns"), or even other aggregators (linked with the relation "has"). An aggregator is possibly related to his icon (i.e. to an aggrsymbol object) by means of the "hasIcon" relation. An aggrsymbol may also "own" some accessory objects, that have to correspond to some of the ones "owned" by the associated aggregator. A classifier object is linked to the classified set of aggregator objects by means of the "collects" relation.
In order to avoid the generation of "unnatural" virtual scenes with either too many or too few virtual world objects, every classifier object has got two attributes (Min and Max), delimiting the range of occurrences.
The kind of data that an Accessory object can support is stated by the "typeOfData" entity, linked by means of the "isOf" relation.

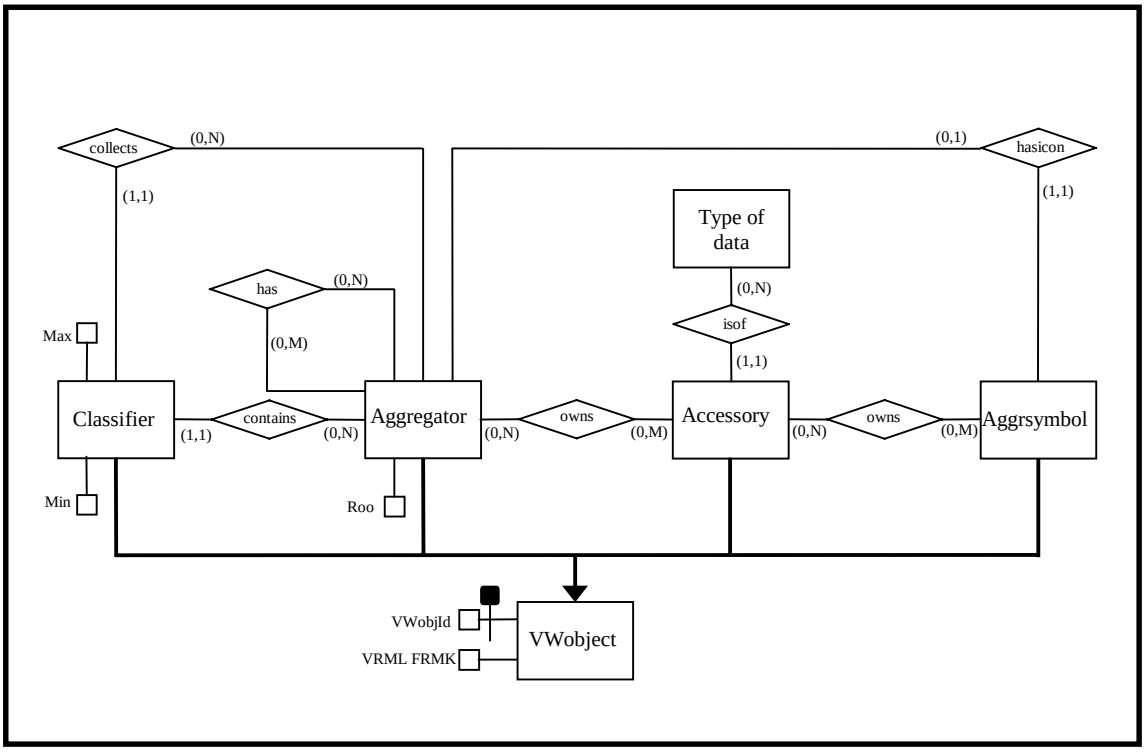


Figure 4.4 - Virtual World Object Repository

The VW objects mentioned in [Massari et al. 96] include also another kind of objects, namely "*Jumpers*", which may be used to switch from the current virtual world to another. Such kinds of objects have not been considered during the implementation of this version of the MDT, since there are still some open questions about their effective meaning and utilisation. For instance, it is not clear if the virtual world to jump at should contain the whole starting query or only a part of it. Moreover, it would not be easy to presume the user's reaction and behaviour if the initially chosen metaphor would be substituted by another one.

The virtual world considered in the example of section 4.1 may be described by a graph like the one displayed in figure 4.5. The starting object (called "Room") is of kind aggregator and represents the whole virtual world. It is linked by means of the "owns" relation to an accessory object (the "Poster") which is capable of displaying an image. The aggregator object "Room" also "contains" a classifier object called "Chest of drawers". It classifies a set of aggregator objects called "Drawer" (see the "collects" relation). Every drawer "owns" an accessory object ("DrLabel") which may display a short text. This accessory is also part of the aggrsymbol object "Drawer front" that identifies every drawer in its closed position (linked by the "hasIcon" relation). The classifier object "Drawer Inside" may "collect" a set of folders. Each folder is capable of displaying some text.

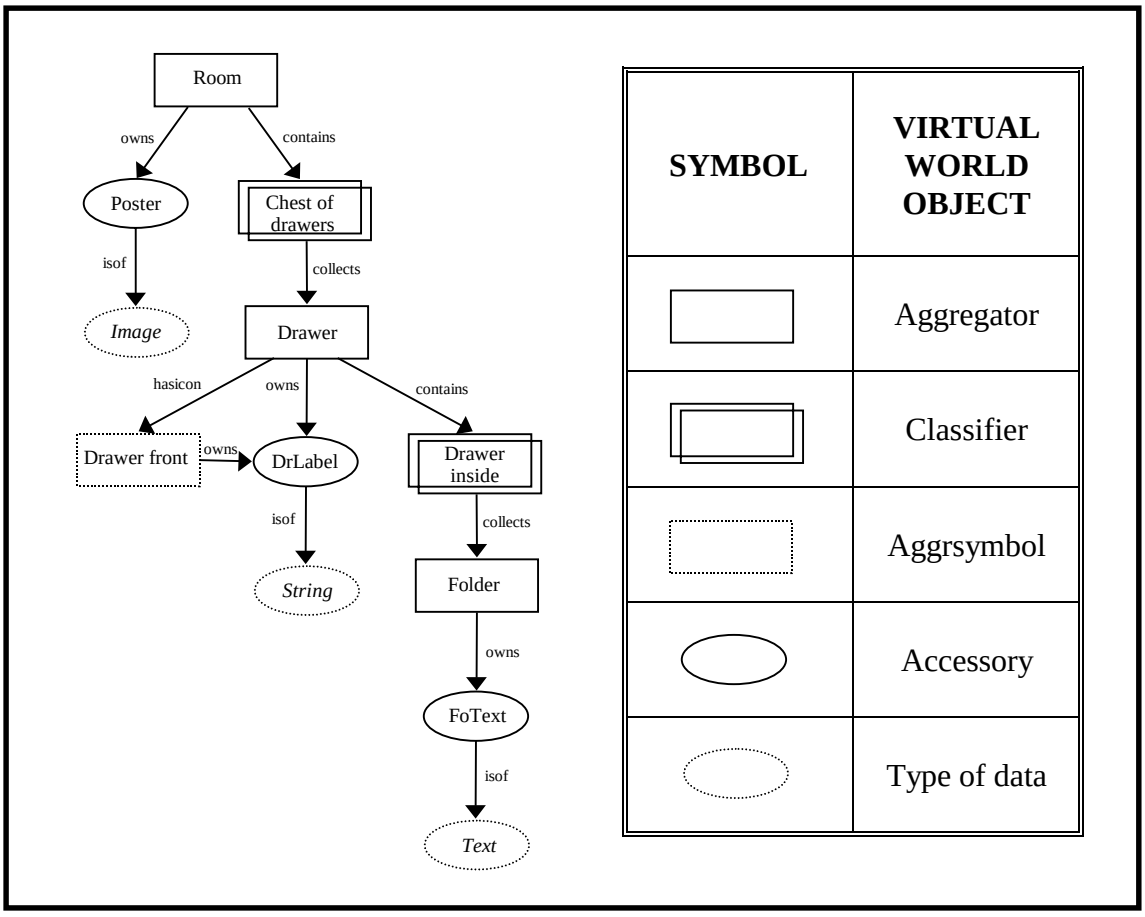


Figure 4.5 - Virtual world "Room"

4.4 Metaphor Repository

Once all possible mappings have been achieved and some of them are found to be suitable for the final result visualisation, the MDT stores its output into the Metaphor Repository (MR). This is achieved by generating a metaphor graph [Massari et al. 96] for each successful mapping. Each metaphor graph has a root node, which will later be provided as a parameter to the Scene Constructor in order to build the final scenes. Its overall structure will correspond to the one of the related structure tree. The information stored in its nodes will be a reference to the corresponding structure tree node and a reference to the mapped virtual world object. Moreover, it is necessary to add nodes for each aggrsymbol that may represent an aggregator from the "outside", i.e. from a different virtual scene. The aggrsymbol nodes of a metaphor graph should also contain a reference to the accessory objects they may support.

The final construction of the virtual scenes will be realised as a scanning of the metaphor graph. The sequence used to traverse the nodes will depend on the user's decisions and needs. Therefore, the scene constructor will only build and visualise the scenes that the user actually wants to visit.

In this document a final version of the metaphor repository will not be given, since its realisation strongly depends on the implementation of the scene constructor algorithm. Anyway a possible metaphor graph for the considered example is shown in section 5, after the specification of the mapping process itself.

5. THE MAPPING PROCESS

The process of searching among all available virtual-world objects, the ones that may be suitable for supporting the query result data-set, is called the mapping process. A set of tools have been evaluated in order to find a proper one for performing this task. The Prolog programming language has been found suitable because it provides a means for representing the structure trees and the virtual-world object structures in a standard format [Bratko 90]. Thus, both query and virtual world are described as lists of relations. The entries of these lists have the following form:

name_of_relation (source_node, destination_node).

A comparison between the list for the query and the list for the virtual-world object can be achieved with a logic-based approach. If we assume the lists of all available virtual worlds to be the starting set of clauses of a Prolog program, we can easily set the query list as the main goal to reach. This process is explained in the next sections.

5.1 The Virtual World list

Every time a new virtual world is added to the VWOR, a new list of relations has to be created. Figure 5.1 contains an example which shows the list related to the virtual world “Room”. As you can notice, the names of every virtual world object contained in the list start with a lowercase letter, in order to be considered by the Prolog interpreter as constant values.
Furthermore, every list has to be completed with other relations, namely the "HasMin" and "HasMax" relations, which link every classifier of the virtual world to their Min and Max attributes.
A "hasIcon" relation will also be inserted for all those aggregator objects which have no Aggrsymbols related. They will be linked to the constant "nil".

```
                    /* VIRTUAL WORLD ROOM */

contains(aggregatorroom,              classifierchestdrawers).
contains(aggregatordrawer,            classifierdrawerinside).

collects(classifierchestdrawers,      aggregatordrawer).
collects(classifierdrawerinside,      aggregatorfolder).

hasmin(classifierchestdrawers,        0).
hasmin(classifierdrawerinside,        1).
hasmax(classifierchestdrawers,        20).
hasmax(classifierdrawerinside,        10).

owns(aggregatorroom,                  accessoryposter).
owns(aggregatorfolder,                accessoryfotext).
owns(aggregatordrawer,                accessorydrlabel).
owns(aggrsymboldrawerfront,           accessorydrlabel).

isof(accessorydrlabel,                typeofdatastring).
isof(accessoryfotext,                 typeofdatatext).
isof(accessoryposter,                 typeofdataimage).

hasicon(aggregatordrawer,             aggrsymboldrawerfront).
hasicon(aggregatorfolder,             nil).
hasicon(aggregatorroom,               nil).
```

Figure 5.1 - Prolog list for virtual world "Room"

5.2 The Query list

A query list is created every time a final user accesses the system with a new query. The names of the structure tree nodes appearing in each relation now begin with an uppercase letter, so that they will be treated as variables. A query list always starts with the instance of variables containing the cardinalities of set_of nodes (see the query "Sting" -list in Figure 5.2). These data will be retrieved by the QT and are available to the system in form of metadata [Massari et al. 96]. The second part of the query list represents the query itself. "*HasMin*" and "*HasMax*" relations are also inserted as well as instructions for controlling the cardinality values. Finally "*hasIcon*" relations are inserted for every record node in order to find out the icons that could possibly represent the record in the virtual scenarios. The third part of the query list manages the output of the variables contents to a file, by means of "write" commands.

```
                /*    Prolog goal list from query 'STING'   */

main:-
                      /*  Set_of nodes Cardinalities */
     CDsMin is 1,
     CDsMax is 8,
     SongsMin is 1,
     SongsMax is 10,

                    /*  Clauses and Cardinality Tests */
          hasicon(RecordSting,              RecordStingIcon),
             owns(RecordSting,              AttributePhoto),
               isof(AttributePhoto,         typeofdataimage),
             contains(RecordSting,          SetOfCDs),
               collects(SetOfCDs,           RecordCD),
                    hasicon(RecordCD,       RecordCDIcon),
               hasmin(SetOfCDs, CMin), CDsMin >= CMin,
               hasmax(SetOfCDs, CMax), CMax >= CDsMax,
                 owns(RecordCD,             AttributeCDTitle),
                   isof(AttributeCDTitle,   typeofdatastring),
                 contains(RecordCD,         SetOfSongs),
                   collects(SetOfSongs,     RecordSong),
                        hasicon(RecordSong, RecordSongIcon),
                  hasmin(SetOfSongs, SoMin), SongsMin >= SoMin,
                  hasmax(SetOfSongs, SoMax), SoMax >= SongsMax,
                    owns(RecordSong,        AttributeSongTitle),
                     isof(AttributeSongTitle, typeofdatastring),

                        /*   File Output   */
     tell(mdtsting),
     write('THE RECORD STING HAS BEEN MAPPED WITH         : '),
     write(RecordSting), nl,
     write('ITS ICON IS                                   : '),
     write(RecordSingerIcon),nl,nl,
     write('THE ATTRIBUTE PHOTO HAS BEEN MAPPED WITH      : '),
     write(AttributePhoto), nl,
     write('THE SET OF CDS HAS BEEN MAPPED WITH           : '),
     write(SetOfCDs), nl,
     write('   MIN WORLD : '),write(CMin),
     write('   MIN QUERY : '),write(CDsMin),nl,
     write('   MAX WORLD : '),write(CMax),
     write('   MAX QUERY : '),write(CDsMax),nl,nl,
     write('THE RECORD CD HAS BEEN MAPPED WITH            : '),
     write(RecordCD), nl,
     write('ITS ICON IS                                   : '),
     write(RecordCDIcon),nl,nl,
     write('THE ATTRIBUTE CD TITLE HAS BEEN MAPPED WITH   : '),
     write(AttributeCDTitle), nl,
     write('THE SET OF SONGS HAS BEEN MAPPED WITH         : '),
     write(SetOfSongs), nl,
     write('   MIN WORLD : '),write(SoMin),
     write('   MIN QUERY : '),write(SongsMin),nl,
     write('   MAX WORLD : '),write(SoMax),
     write('   MAX QUERY : '),write(SongsMax),nl,nl,
     write('THE RECORD SONG HAS BEEN MAPPED WITH          : '),
     write(RecordSong), nl,
     write('ITS ICON IS                                   : '),
     write(RecordSongIcon),nl,nl
     write('THE ATTRIBUTE SONG TITLE HAS BEEN MAPPED WITH : '),
     write(AttributeSongTitle), nl,nl,
     write('##############################################################################'),
     nl,nl,fail.
```

Figure 5.2 - Prolog goal list from query "Sting"

5.3 Executing the process

A command file, as the one shown in Figure 5.3, will be generated every time a mapping process is invoked. It contains all basic steps to be carried out by the Prolog interpreter. First, a file called "multi" is loaded in order to inform the system that some clauses could be used in several different files. The second step consists of the loading of the virtual world lists. In our example we have only one virtual world to load (the "Room"). The *Virgilio* system administrator or some other tools will update this part of the command file by inserting the names of all the virtual world lists that have to be considered for the mapping. In this way it is possible to take into account some constraints (for instance the user model) for the creation of efficient metaphors.
After the loading of a query list (see the instruction "[Sting]." in our example) a final instruction ("main.") will tell the system to begin the searching of all possible mappings. This is realised by a built-in backtracking algorithm which is the standard way to proceed by every Prolog interpreter.
Following a depth first search strategy, the system makes a comparison between the relational structure of the query list and the relational structure of the virtual world list. A set_of node will be mapped to a classifier object only if the comparison between the related cardinalities does not fail. An attribute node will be mapped to an accessory object only if they support the same type of data. In case of success the variables appearing in the query list are instanced to the constants of the virtual world list, and saved to a file for further uses. For the generation of a metaphor graph, information about aggrsymbols are also added to the output file.

```
/*  COMMAND FILE FOR THE MAPPING OF QUERY "STING"   */

          /*  LOADING OF PREFERENCES   */
                [multi].

          /*  LOADING OF VIRTUAL WORLDS   */
                [room].


          /*   LOADING OF THE QUERY   */
                [sting].

          /*   GOAL TO REACH   */
                main.
```

Figure 5.3 - Command File for the mapping process of query "Sting"

Figure 5.4 shows the output of the mapping process applied to the example of the query "Sting". After the definition of the final scene constructor algorithm, the output of the processing step will be expanded with all the necessary data describing the involved objects. For example, it will be necessary to store references to the virtual world objects and the structure tree nodes, which are part of the resulting metaphor graph.

```
          /*   MAPPING RESULTS FOR QUERY "STING"   */

THE RECORD STING HAS BEEN MAPPED WITH         : aggregatorroom
ITS ICON IS                                   : nil

THE ATTRIBUTE PHOTO HAS BEEN MAPPED WITH      : accessoryposter
THE SET OF CDS HAS BEEN MAPPED WITH           : classifierchestdrawers
   MIN WORLD : 0   MIN QUERY : 1
   MAX WORLD : 20   MAX QUERY : 8

THE RECORD CD HAS BEEN MAPPED WITH            : aggregatordrawer
ITS ICON IS                                   : aggrsymboldrawerfront

THE ATTRIBUTE CD TITLE HAS BEEN MAPPED WITH   : accessorydrawerlabel
THE SET OF SONGS HAS BEEN MAPPED WITH         : classifierfolders
   MIN WORLD : 1   MIN QUERY : 1
   MAX WORLD : 10   MAX QUERY : 10

THE RECORD SONG HAS BEEN MAPPED WITH          : aggregatorfolder
ITS ICON IS                                   : nil

THE ATTRIBUTE SONG TITLE HAS BEEN MAPPED WITH : accessoryfotext

##########################################################################
```

Figure 5.4 - Mapping results for query "Sting"

A possible metaphor graph would be the one of figure 5.5. One can see that its general structure corresponds to that of the related structure tree.

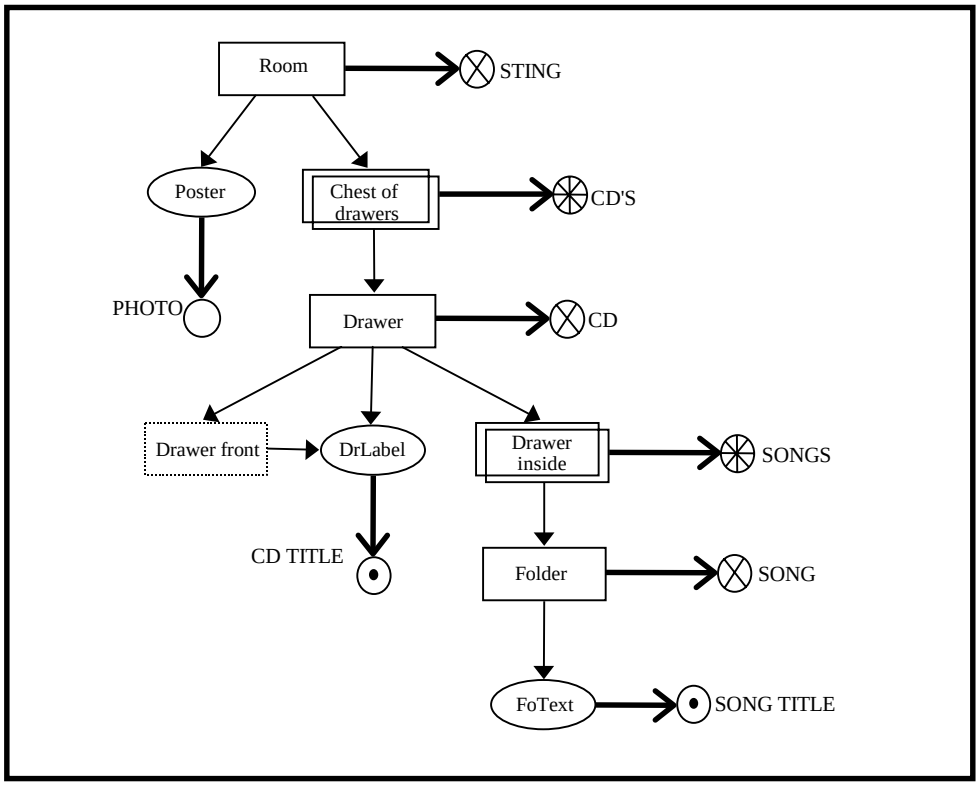


Figure 5.5 - Metaphor Graph for query "Sting". The thick edges represent the generated mappings.

6. FUTURE WORK

In order to achieve compatibility among the MDT and the other modules of *Virgilio* (see Figure 2.1) a set of interfaces should be implemented. These interfaces have been shown in Figure. 4.1. and explained in Section 4.

After the realisation of a first simple scene constructor algorithm, it would be possible to integrate these components, and achieve the first generic version of the *Virgilio* system. It would be of course necessary to create some more virtual worlds to store into the VWOR. Figure 6.1 shows a possible data flow chart for the *Virgilio* version 1.0. The final user formulates his query by means of a visual query interface. The output of this step is an SQL script of the query. It will be processed by the Structure Tree Generator component and stored in form of a structure tree into the QR. The List Generator component will now produce the query list to be used within the MDT core. The lists of all available virtual worlds will also be furnished, and after the processing step, all possible mappings will be available in form of metaphor graphs in the MR. Finally, the user will interact again with the scene constructor, choosing the most appropriate mapping and browsing at last the requested data.

Please note that all the processing steps explained above can be carried out on-line and may be also available over the web. Moreover, since there will be a low number of available virtual worlds at the beginning, the mapping process won't cost too much execution time.

It is clear that this first *Virgilio* release does not take into account all of the features explained in the previous sections. For instance, the metaphor effect of the produced mappings is not verified, and the queries and mappings of preceding sessions (stored in the QR or the MR) are not considered.
By the way, *Virgilio* 1.0 could be used for special purposes in which the available virtual worlds are fixed and addressed to a selected user group (for example the visitors of a trade fair).

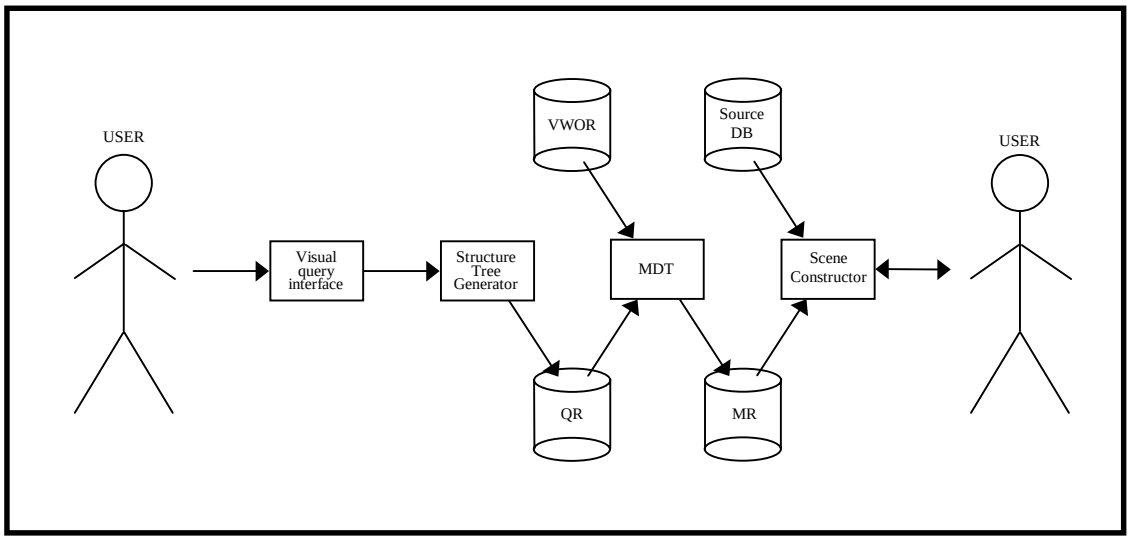


Figure 6.1 - Data flow chart for *Virgilio* version 1.0

7. REFERENCES

[Aloia et al. 96] Aloia, N., Matera, M., Paternò, F. (1996)
A Semantics-based Approach to Designing Presentations for Multimedia Database Query Results, Proceedings of AVI 96
Gubbio, Italy.

[Atzeni and De Antonellis 93] Atzeni P., De Antonellis V., (1993)
Relational Database Theory, Benjamin Cummings.

[Carroll et al. 88] Carroll, J.M., Mack, R.L., Kellogg W.A. (1988)
Interface Metaphors and user interface Design, Chapter 3 of the Handbook of Human-Computer Interaction, Elsevier
Science Publishers B.V

[Catarci and Costabile 95] Catarci, T., Costabile, M.F., (1995)
Special Issue on Visual Query Systems
Journal of Visual Languages and Computing Vol. 6

[Chang et al. 93] Chang, S.K., Costabile, M.F., Levialdi, S., (1993)
Modelling users in an adaptive visual interface for database systems, Journal of Visual Languages and computing, vol. 4.

[Costabile et al. 95] Costabile, M.F., Catarci, T., Matera, M. (1995)
Visual Metaphors for interacting with Databases,
SIGCHI Bulletin, Volume 27, No. 2

[Erickson 90] Erickson, T.D. (1990)
Working with Interface Metaphors
In The Art of Human-Computer Interface
Design, Addison-Wesley, Reading, Mass.

[Gentner 80] Gentner, D. (1980)
The Structure of Analogical Models in Science,
BBN Technical Report, No. 4451.

[Haber et al. 94] Haber, E.M., Ioannidis, Y.E., Livny, M. (1994)
Foundations of Visual Metaphors for Schema Display,
Journal of Intelligent Information Systems, Vol. 3.

[Lakoff and Johnson80] Lakoff, G., Johnson, M. (1980)
Metaphors we live by,
The university of Chicago Press

[Marcus 94] Marcus, A. (1994)
Managing Metaphors for Advanced User Interfaces,
Proceedings of AVI 94, Bari, Italy.

[Martin 90] Martin, J.H. (1990)
A Computational Model of Metaphor Interpretation,
Academic Press.

[Massari et al. 95] Massari, A., Pavani, S., Saladini, L., Chrysantis, P.K. (1995)
Query by icons in Proceedings of the International
Conference ACM-SIGMOD, S. Jose, California, USA.

[Massari et al. 96] Massari, A., Saladini, L., Hemmje, M. (1996)
Architecture and System Specification of Virgilio
appearing in GMD Arbeitsberichte, Gesellschaft fuer Mathematik und Datenverarbeitung, Darmstadt, Germany.

[Massari et al. 97] Massari, A., Saladini, L., Hemmje, M., Sisinni, F. (1997)
Virgilio: a non-immersive VR system to browse multimedia databases, Proceedings of the IEEE International Conference on multimedia computing systems, IEEE computer society press, 573-580

[Massari et al. 98] Massari, A., Saladini, L., Hemmje, M., Sisinni, F., Paradiso, A., Napolitano, W., Leissler, M.
Virtual Reality Systems for browsing multimedia
To appear in Furth, B. : Handbook of Multimedia computing

[Paradiso and Hemmje 96] Paradiso, A., Hemmje, M., (1996)
A generic refinement of the Virgilio System's design and a prototypical architecture, in GMD Arbeitspapiere No. 1093
Gesellschaft fuer Mathematik und Datenverarbeitung,
Darmstadt, Germany.

[Bratko 90] Bratko, I. (1990)
Prolog, Programming for artificial intelligence,
Addison Wesley Publishing

[VRML] Virtual Reality Modelling Language
Spec. On www. Http://vrml.sgi.com/moving-worlds/spec

8. Acknowledgements

The present paper was written during Marcello L'Abbate's stay as a student assistant at the Institute for Integrated Publication and Information Systems (IPSI) of GMD – German National Research Center for Information Technology in Darmstadt and at the University of Bari.

It is the result of requirements that were gained during the earlier development of the first prototypes of the *Virgilio* system. Therefore, it is to be seen, on the one hand, within the framework of the *Virgilio* project which has been co-operatively conducted by the Department for Visual Interaction Tools (VISIT) of GMD-IPSI, the University of Rome "La Sapienza" and the University of Bari in Italy. On the other hand, the work is based on discussions led with colleagues about related implementations and publications within the European Working Group for Foundations of Threedimensional Information Visualisation (FADIVA). Therefore, it is also a self-contained and independent publication.

In particular, we would like to thank Maria Francesca Costabile, Donato Malerba, Aldo Paradiso, Annabella Loconsole for many interesting discussions and their support during the implementation of the MDT component.

APPENDIX A - Mapping Example I

The query used by the first *Virgilio* prototype [Paradiso and Hemmje 96], dealing with music types and authors, is called “Music”.

Query “Music”: *Retrieve all the names of the music types stored in the database. For each music type retrieve a short text, which explains its historical background. For each music type retrieve the name and the picture of the bands typically performing such a music type. For each band retrieve all the names and cover images of the albums released by the band. For each album retrieve the titles of the songs contained in it.*

Its structure tree is shown in Figure A.1. One can identify that the first node is a record node which represents the whole query. It has only one child, namely a set_of node, which contains the SQL script for a part of the first sentence of the query text ("retrieve all the music types stored in the database"). A single music type appears in the structure tree as a record node, which "owns" the requested information (name and short text, see query text) in form of attribute nodes. It also contains another set_of node, which represents the set of all the bands performing a certain music type. The rest of the structure tree can now be easily traced, because it contains repetitions of the above examined notions.

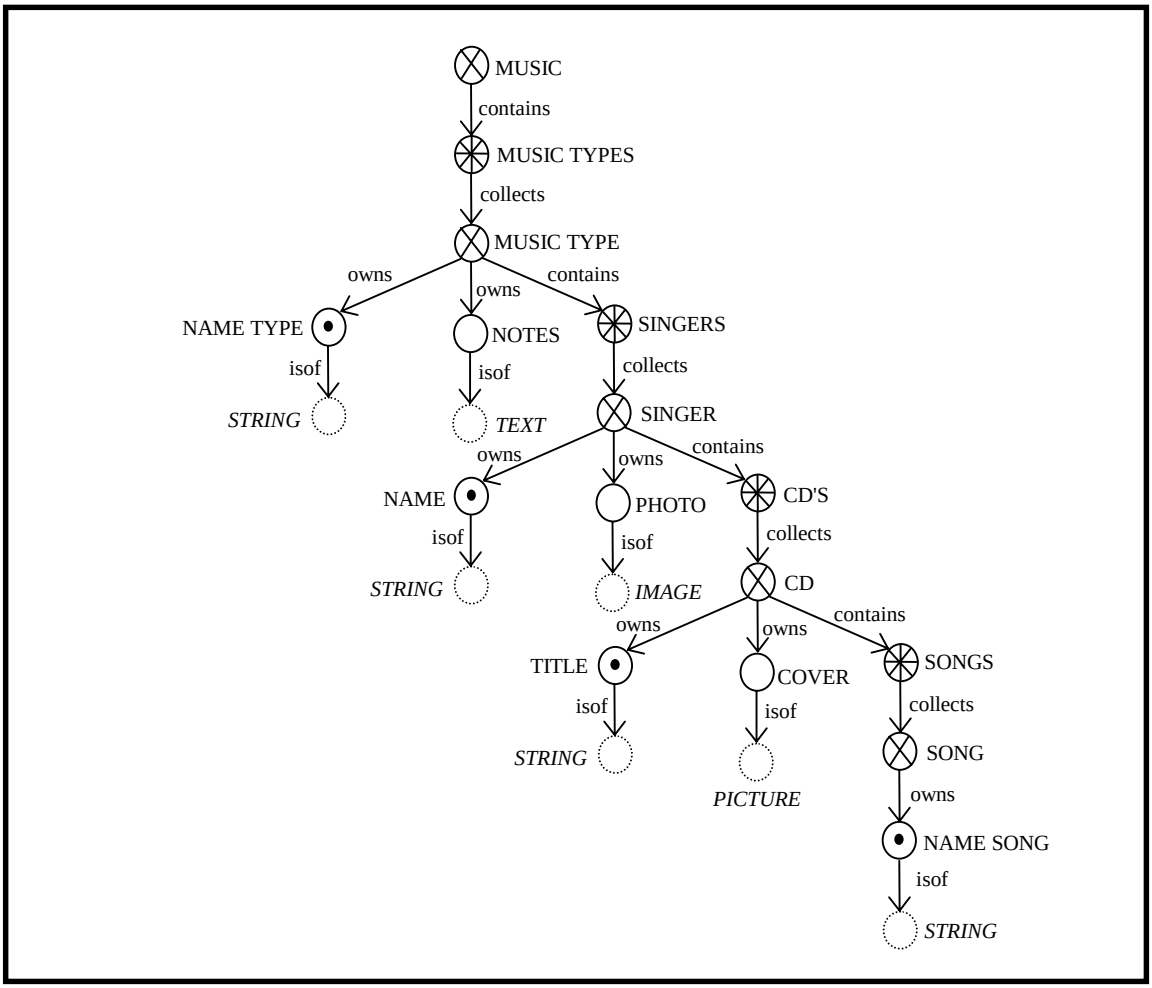


Figure A.1 - Structure Tree representing the query "Music"

Every node of the structure tree contains a short SQL query, determined by the Structure Tree Generator. As an example, consider the query script contained in the classifier structure tree node "singers":

```
SELECT    *
FROM    singers
WHERE   name IN   (SELECT   singers.name
                   FROM     tipically_sings
                   WHERE    music_name =  <selected_music_type> );
```

The result of this SQL sentence will be a list containing all the names of the singers (stored in the source DB) whose released music may belong to a specific music type, chosen in a previous step during the browsing of the result (set in the "selected_music_type" variable).

Another example is the query script of the attribute node "photo":

```
SELECT   photo
FROM     singers
WHERE   ( name IN ( SELECT   singers.name
                    FROM     tipically_sings
                    WHERE    music_name =  <selected_music_type>))
          AND   ( name = <selected_singer_name> );
```

The "selected_singer_name" variable contains the name of the singer whose photo is going to be retrieved and visualised.

The virtual world considered for this example also appears in the *Virgilio* prototype. You may follow its structure in Figure A.2. The table on figure A.2b explains the meaning of the objects contained in the graph of figure A.2. The schema describes a building with a main entrance hall leading to an elevator. This Classifier object gives the possibility to access an Aggregator object called "floor" by pushing on its aggrsymbol, namely the button on a button table. Every button has an accessory associated to it (a button label), which identifies the floor to reach. The floor contains a sign (the accessory *fname*), which displays the same information as the previously pushed button. It also may contain two different kinds of sideboards. The first (a simple accessory) capable of displaying a text and the second (an aggregator called *complex board*) may display a text as well as a picture. A floor aggregates also a corridor with rooms, which may be entered by opening a door with a label on it (the aggrsymbol of aggregator room). In a room you may find an Accessory poster with an image and two other classifiers, namely a chest of drawers and a photo album on a table (see Figure A.2).

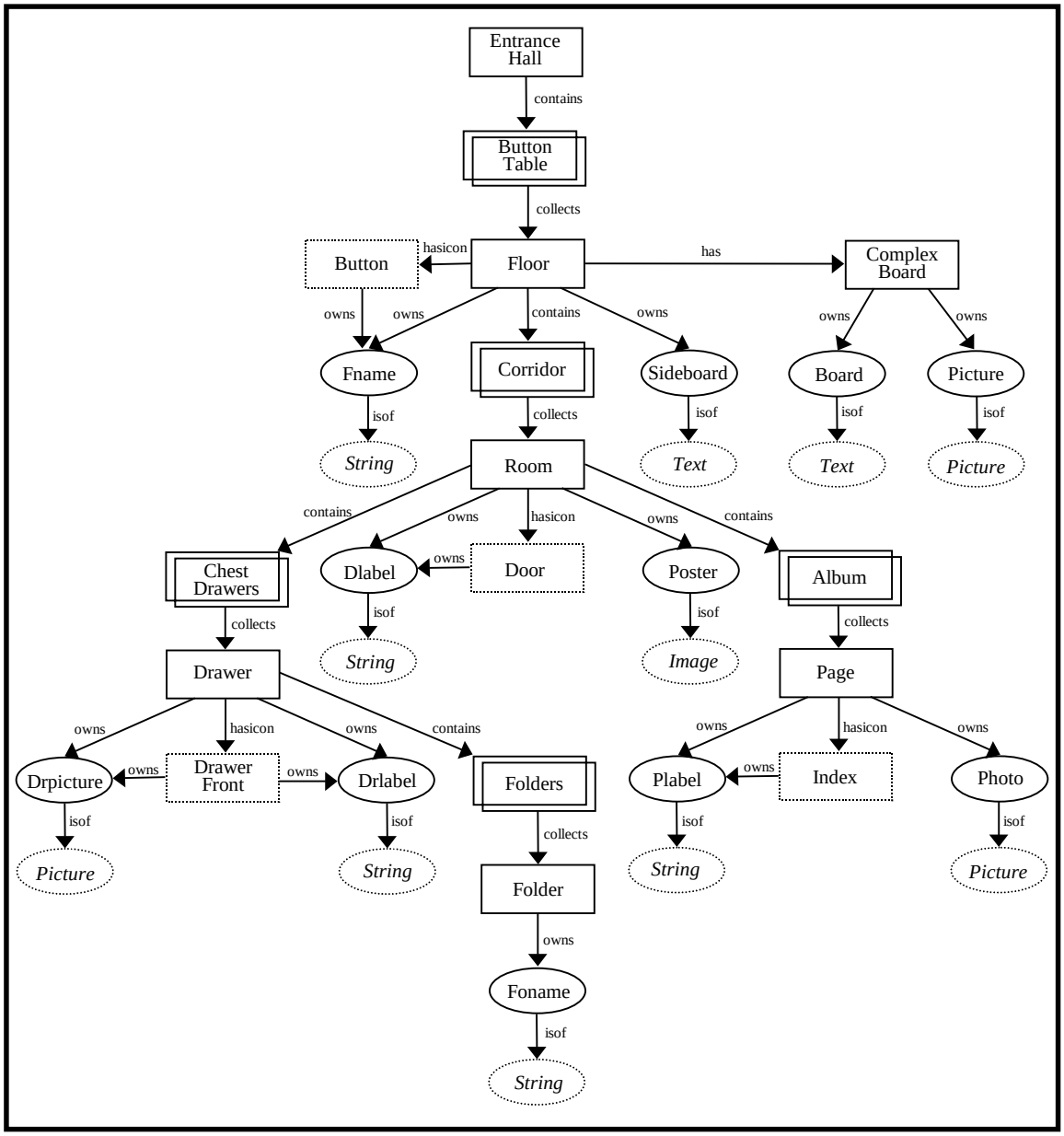


Figure A.2 - Virtual world "Building" structure graph

VW OBJECT	NODE TYPE	MEANING
Entrance Hall	Aggregator	Main Entrance Hall to the Building
Button Table	Classifier	Elevator and Button Table
Button	Aggrsymbol	Button on the Button Table of the Elevator
Floor	Aggregator	Generic Floor of the Building
Fname	Accessory	Name of a Floor
Sideboard	Accessory	Board with a text
Complex Board	Aggregator	Board with text and a picture
Board	Accessory	Text on the Complex Board
Picture	Accessory	Picture on the Complex Board
Corridor	Classifier	Corridor of Rooms
Door	Aggrsymbol	Door to access a Room
Room	Aggregator	Room of the Building
Dlabel	Accessory	Label on a Door
Poster	Accessory	Picture on the wall of a Room
Album	Classifier	Book on a table in the Room
Index	Aggrsymbol	Summary of the contents of the Book
Page	Aggregator	Single Page of the Book
Plabel	Accessory	Name of a Page
Photo	Accessory	Picture on a Page
Chest Drawers	Classifier	Piece of Furniture with Drawers
Drawer Front	Aggrsymbol	Front Panel of a Drawer
Drawer	Aggregator	Drawer from the Chest
Drlabel	Accessory	Label on a Drawer front
Drpicture	Accessory	Picture on a Drawer front
Folders	Classifier	Set of Folders in a Drawer
Folder	Aggregator	Folder in the Drawer
Foname	Accessory	Name of the Folder

Figure A.2b Meaning of the objects contained in the graph of figure A.2.

The following figure A.3 shows the Prolog list related to the virtual world "Building". This virtual world contains seven aggregator objects, and therefore seven hasicon relations can be found on the Prolog list. Three of them are linked to the constant nil because they do not have a related aggrsymbol object.
It is of course possible to expand a virtual world by adding more objects. In this case it will be necessary to update the virtual world list by inserting the new relations.

```
            /* VIRTUAL WORLD BUILDING */


contains(aggregatorentrancehall, classifierbuttontable).
contains(aggregatorroom,      classifierchestdrawers).
contains(aggregatorroom,      classifieralbum).
contains(aggregatorfloor,     classifiercorridor).
contains(aggregatordrawer,    classifierfolders).

collects(classifierbuttontable,  aggregatorfloor).
collects(classifiercorridor,  aggregatorroom).
collects(classifierchestdrawers, aggregatordrawer).
collects(classifierfolders,   aggregatorfolder).
collects(classifieralbum,     aggregatorpage).

hasmin(classifierbuttontable, 2).
hasmin(classifiercorridor,    1).
hasmin(classifierchestdrawers, 0).
hasmin(classifierfolders,     1).
hasmin(classifieralbum,       1).

hasmax(classifierbuttontable, 20).
hasmax(classifiercorridor,    30).
hasmax(classifierchestdrawers, 20).
hasmax(classifierfolders,     10).
hasmax(classifieralbum,       30).

owns(aggregatorfloor,         accessoryfname).
owns(aggregatorfloor,         accessorysideboard).
owns(aggregatorroom,          accessoryposter).
owns(aggregatorcomplexboard,  accessoryboard).
owns(aggregatorcomplexboard,  accessorypicture).
owns(aggregatorfolder,        accessoryfoname).
owns(aggrsymboldoor,          accessorydlabel).
owns(aggregatorroom,          accessorydlabel).
owns(aggrsymboldrawerfront,   accessorydrpicture).
owns(aggrsymboldrawerfront,   accessorydrlabel).
owns(aggrsymbolbutton,        accessoryfname).
owns(aggrsymbolindex,         accessoryplabel).
owns(aggregatordrawer,        accessorydrlabel).
owns(aggregatordrawer,        accessorydrawerpicture).
owns(aggregatorpage,          accessoryplabel).
owns(aggregatorpage,          accessoryphoto).

isof(accessoryfname,          typeofdatastring).
isof(accessorydrlabel,        typeofdatastring).
isof(accessoryfoname,         typeofdatastring).
isof(accessoryplabel,         typeofdatastring).
isof(accessorydlabel,         typeofdatastring).
isof(accessorysideboard,      typeofdatatext).
isof(accessoryboard,          typeofdatatext).
isof(accessorypicture,        typeofdatapicture).
isof(accessorydrpicture,      typeofdatapicture).
isof(accessoryphoto,          typeofdatapicture).
isof(accessoryposter,         typeofdataimage).

has(aggregatorfloor,          aggregatorcomplexboard).

hasicon(aggregatorroom,       aggrsymboldoor).
hasicon(aggregatordrawer,     aggrsymboldrawerfront).
hasicon(aggregatorpage,       aggrsymbolindex).
hasicon(aggregatorfloor,      aggrsymbolbutton).
hasicon(aggregatorcomplexboard, nil).
hasicon(aggregatorentrancehall, nil).
hasicon(aggregatorfolder,     nil).
```

Figure A.3 - Prolog list for virtual World "Building"

The query repository interface (refer to section 4.1) will generate the query list displayed in figures A.4a and A.4b. The first part of the list contains the instance of the classifier cardinalities (see section 5.2). The second part of the list represents the query itself.
The *Virgilio* version 1.0 will discard this list after achieving the mapping. Future versions may keep every structure tree and related list, in order to be utilised in more and successive sessions.

```
            /*    Prolog goal list from query 'MUSIC'   */


main:-

       /*  Classifier Cardinalities */

        MusicTypesMin is 5,
        MusicTypesMax is 15,
        SingersMin is 2,
        SingersMax is 30,
        CDsMin is 1,
        CDsMax is 8,
        SongsMin is 1,
        SongsMax is 10,


        /*  Clauses and Cardinality Tests */

        contains(RecordMusic, SetOfMusicTypes),
        hasicon(RecordMusic, RecordMusicIcon),
          collects(SetOfMusicTypes, RecordMusicType),
            hasicon(RecordMusicType, RecordMusicTypeIcon),
         hasmin(SetOfMusicTypes, MTMin), MusicTypesMin >= MTMin,
         hasmax(SetOfMusicTypes, MTMax), MTMax >= MusicTypesMax,
            owns(RecordMusicType, AttributeNameType),
              isof(AttributeNameType, typeofdatastring),
            owns(RecordMusicType, AttributeNotes),
              isof(AttributeNotes, typeofdatatext),
            contains(RecordMusicType, SetOfSingers),
              collects(SetOfSingers, RecordSinger),
                  hasicon(RecordSinger, RecordSingerIcon),
               hasmin(SetOfSingers, SiMin), SingersMin >= SiMin,
               hasmax(SetOfSingers, SiMax), SiMax >= SingersMax,
                  owns(RecordSinger, AttributeName),
                    isof(AttributeName, typeofdatastring),
                  owns(RecordSinger, AttributePhoto),
                    isof(AttributePhoto, typeofdataimage),
                  contains(RecordSinger, SetOfCDs),
                    collects(SetOfCDs, RecordCD),
                         hasicon(RecordCD, RecordCDIcon),
                   hasmin(SetOfCDs, CMin), CDsMin >= CMin,
                   hasmax(SetOfCDs, CMax), CMax >= CDsMax,
                      owns(RecordCD, AttributeTitle),
                        isof(AttributeTitle, typeofdatastring),
                      owns(RecordCD, AttributeCover),
                        isof(AttributeCover, typeofdatapicture),
                      contains(RecordCD, SetOfSongs),
                        collects(SetOfSongs, RecordSong),
                              hasicon(RecordSong, RecordSongIcon),
                         hasmin(SetOfSongs, SoMin), SongsMin >= SoMin,
                         hasmax(SetOfSongs, SoMax), SoMax >= SongsMax,
                            owns(RecordSong, AttributeNameSong),
                              isof(AttributeNameSong, typeofdatastring),
```

Figure A.4a - First and second part of the query list "Music"

Once a mapping has been determined, the variables contents are stored in a file (named "mdtmusic"), in order to be processed by the metaphor repository interface, which will generate a metaphor graph. In this example the output has been designed only for information purposes, so that one can verify the occurred mappings (see figure A.4b).

```
                        /*   File Output   */


        tell(mdtmusic),
        write('THE RECORD MUSIC HAS BEEN MAPPED WITH        : '),
        write(RecordMusic),nl,
        write('ITS ICON IS                                  : '),
        write(RecordMusicIcon),nl,nl,
        write('THE SET OF MUSIC TYPES HAS BEEN MAPPED WITH  : '),
        write(SetOfMusicTypes), nl,
        write('   MIN WORLD : '),write(MTMin),
        write('   MIN QUERY : '),write(MusicTypesMin),nl,
        write('   MAX WORLD : '),write(MTMax),
        write('   MAX QUERY : '),write(MusicTypesMax),nl,nl,
        write('THE RECORD MUSIC TYPE HAS BEEN MAPPED WITH   : '),
        write(RecordMusicType), nl,
        write('ITS ICON IS                                  : '),
        write(RecordMusicTypeIcon),nl,nl,
        write('THE ATTRIBUTE NAME TYPE HAS BEEN MAPPED WITH : '),
        write(AttributeNameType),nl,
        write('THE ATTRIBUTE NOTES HAS BEEN MAPPED WITH     : '),
        write(AttributeNotes), nl,
        write('THE SET OF SINGERS HAS BEEN MAPPED WITH      : '),
        write(SetOfSingers), nl,
        write('   MIN WORLD : '),write(SiMin),
        write('   MIN QUERY : '),write(SingersMin),nl,
        write('   MAX WORLD : '),write(SiMax),
        write('   MAX QUERY : '),write(SingersMax),nl,nl,
        write('THE RECORD SINGER HAS BEEN MAPPED WITH       : '),
        write(RecordSinger), nl,
        write('ITS ICON IS                                  : '),
        write(RecordSingerIcon),nl,nl,
        write('THE ATTRIBUTE NAME HAS BEEN MAPPED WITH      : '),
        write(AttributeName), nl,
        write('THE ATTRIBUTE PHOTO HAS BEEN MAPPED WITH     : '),
        write(AttributePhoto), nl,
        write('THE SET OF CDS HAS BEEN MAPPED WITH          : '),
        write(SetOfCDs), nl,
        write('   MIN WORLD : '),write(CMin),
        write('   MIN QUERY : '),write(CDsMin),nl,
        write('   MAX WORLD : '),write(CMax),
        write('   MAX QUERY : '),write(CDsMax),nl,nl,
        write('THE RECORD CD HAS BEEN MAPPED WITH           : '),
        write(RecordCD), nl,
        write('ITS ICON IS                                  : '),
        write(RecordCDIcon),nl,nl,
        write('THE ATTRIBUTE TITLE HAS BEEN MAPPED WITH     : '),
        write(AttributeTitle), nl,
        write('THE ATTRIBUTE COVER HAS BEEN MAPPED WITH     : '),
        write(AttributeCover), nl,
        write('THE SET OF SONGS HAS BEEN MAPPED WITH        : '),
        write(SetOfSongs), nl,
        write('   MIN WORLD : '),write(SoMin),
        write('   MIN QUERY : '),write(SongsMin),nl,
        write('   MAX WORLD : '),write(SoMax),
        write('   MAX QUERY : '),write(SongsMax),nl,nl,
        write('THE RECORD SONG HAS BEEN MAPPED WITH         : '),
        write(RecordSong), nl,
        write('ITS ICON IS                                  : '),
        write(RecordSongIcon),nl,nl,
        write('THE ATTRIBUTE NAME SONG HAS BEEN MAPPED WITH : '),
        write(AttributeNameSong), nl,
write('################################################################################'),
        nl,nl,fail.
```

Figure A.4b - Third part of the query list "Music"

The command file for the mapping process contains instructions for the loading of the virtual world list ("[building]."), for the loading of the query list ("[music]."), and for the beginning of the mapping process ("main."). It is displayed in figure A.5.

```
/*  COMMAND FILE FOR THE MAPPING OF QUERY "MUSIC"   */

               /*  LOADING OF PREFERENCES   */

                       [multi].

               /*  LOADING OF VIRTUAL WORLDS   */

                       [building].

               /*   LOADING OF THE QUERY   */

                       [music].

               /*   GOAL TO REACH   */

                       main.
```

Figure A.5 - Command file for the mapping

The result of the mapping process (i.e. the contents of the file "mdtmusic") are contained in figure A.6.

```
          /*   MAPPING RESULTS FOR QUERY "MUSIC"   */


THE RECORD MUSIC HAS BEEN MAPPED WITH        : aggregatorentrancehall
ITS ICON IS                                  : nil

THE SET OF MUSIC TYPES HAS BEEN MAPPED WITH  : classifierbuttontable
   MIN WORLD : 2   MIN QUERY : 5
   MAX WORLD : 20   MAX QUERY : 15

THE RECORD MUSIC TYPE HAS BEEN MAPPED WITH   : aggregatorfloor
ITS ICON IS                                  : aggrsymbolbutton

THE ATTRIBUTE NAME TYPE HAS BEEN MAPPED WITH : accessoryfname
THE ATTRIBUTE NOTES HAS BEEN MAPPED WITH     : accessorysideboard
THE SET OF SINGERS HAS BEEN MAPPED WITH      : classifiercorridor
   MIN WORLD : 1   MIN QUERY : 2
   MAX WORLD : 30   MAX QUERY : 30

THE RECORD SINGER HAS BEEN MAPPED WITH       : aggregatorroom
ITS ICON IS                                  : aggrsymboldoor

THE ATTRIBUTE NAME HAS BEEN MAPPED WITH      : accessorydlabel
THE ATTRIBUTE PHOTO HAS BEEN MAPPED WITH     : accessoryposter
THE SET OF CDS HAS BEEN MAPPED WITH          : classifierchestdrawers
   MIN WORLD : 0   MIN QUERY : 1
   MAX WORLD : 20   MAX QUERY : 8

THE RECORD CD HAS BEEN MAPPED WITH           : aggregatordrawer
ITS ICON IS                                  : aggrsymboldrawerfront

THE ATTRIBUTE TITLE HAS BEEN MAPPED WITH     : accessorydrlabel
THE ATTRIBUTE COVER HAS BEEN MAPPED WITH     : accessorydrpicture
THE SET OF SONGS HAS BEEN MAPPED WITH        : classifierfolders
   MIN WORLD : 1   MIN QUERY : 1
   MAX WORLD : 10   MAX QUERY : 10

THE RECORD SONG HAS BEEN MAPPED WITH         : aggregatorfolder
ITS ICON IS                                  : nil

THE ATTRIBUTE NAME SONG HAS BEEN MAPPED WITH : accessoryfoname

#########################################################################
```

Figure A.6 - Mapping results

The metaphor graph resulting from the mapping of query "Music" with virtual world "Building" has got the same overall structure as the query "Music" structure tree. The graph contains only the virtual-world objects which have a corresponding structure tree node mapped to them. Figure A.7 depicts a possible representation of the metaphor graph.

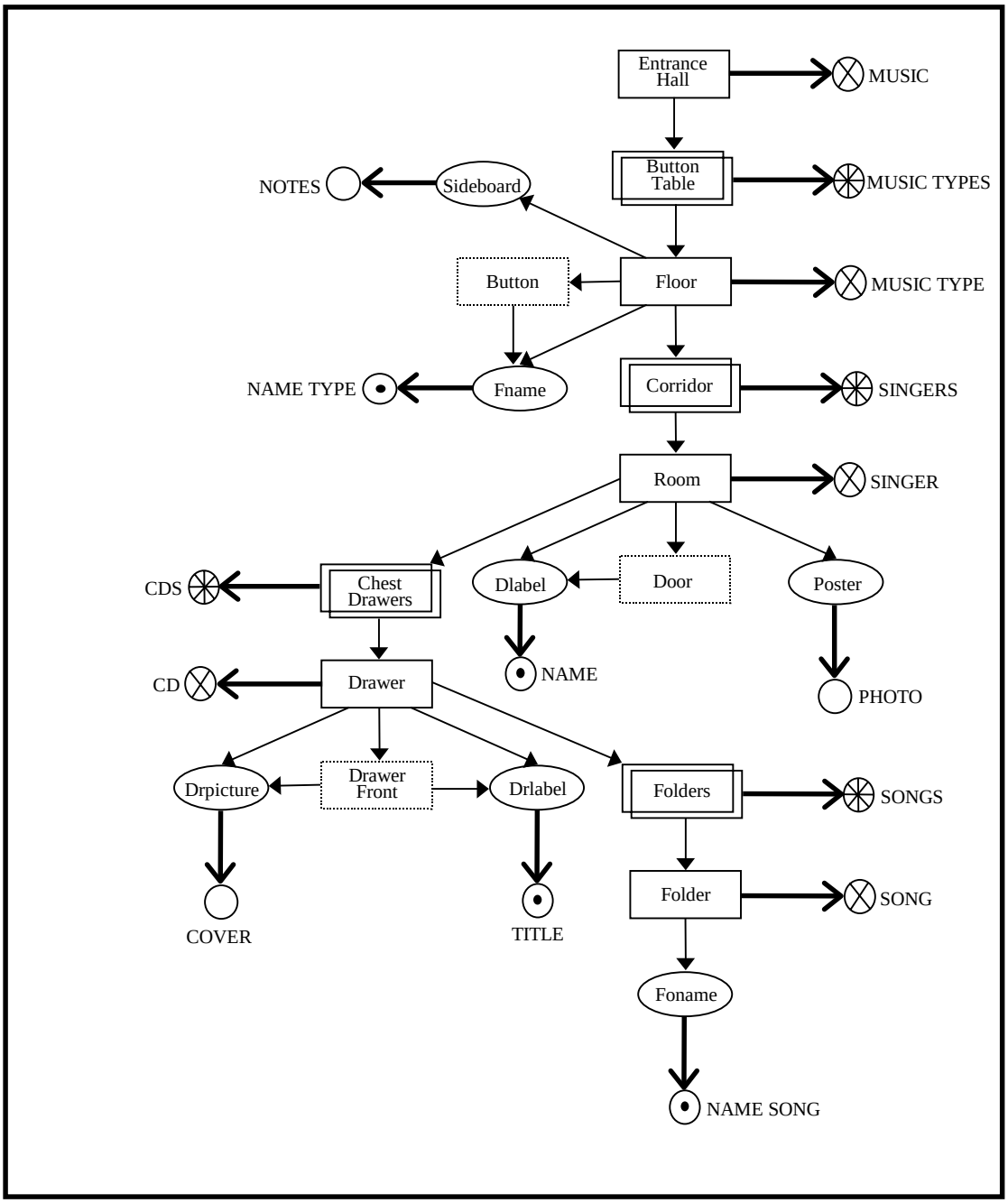


Figure A.7 - Metaphor graph for the mapping of query "Music" with virtual world "Building". The thick edges represent the mappings.

APPENDIX B - Mapping Example II

In the following, another example is treated. It introduces a new query called "Italy" regarding Italian cities and their main tourist attractions. Also a new virtual world “Railway” has been considered, dealing with railway and underground stations. The mapping process is applied to both virtual worlds “Building” and “Railway”, and produces three different mappings. The query “Italy” may be formulated as follows:

Query “Italy”: *Retrieve all the names of the Italian regions.*

For each region retrieve the notes of its historical background and the image of its geographical map.

For each region retrieve the names of its provinces.

For each province retrieve the name and the picture of its main tourist attractions.

Retrieve for each province, moreover, the names of all the towns belonging to its area with a population greater than 30000 people.

For each town retrieve the names of its main tourist attractions.

Its structure tree is shown in figure B.1.

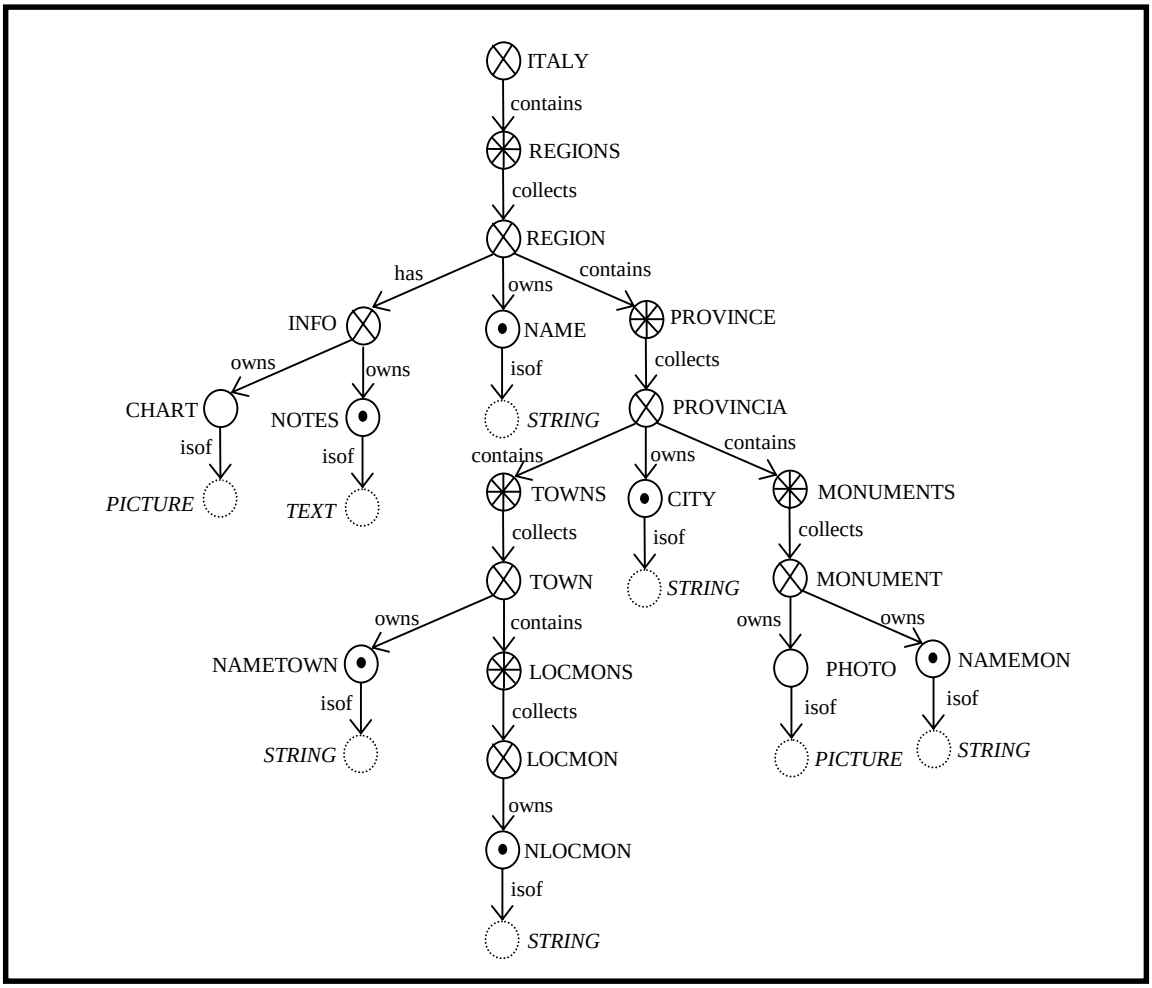


Figure B.1 - Structure Tree query "Italy"

Figure B.2 shows the complete query list for query "Italy". It is divided again into three parts according to the specifications given in section 5.2.

```
   /*    Prolog goal list from query 'ITALY'    */


main:-

/*  Classifier Cardinalities    */

RegionsMin is 20,
RegionsMax is 20,
ProvinceMin is 1,
ProvinceMax is 15,
MonumentsMin is 1,
MonumentsMax is 20,
TownsMin is 1,
TownsMax is 8,
LocMonsMin is 1,
LocMonsMax is 10,




/*  Clauses and Cardinality Tests   */

contains(RecordItaly, SetOfRegions) ,
hasicon(RecordItaly, RecordItalyIcon),
  collects(SetOfRegions, RecordRegion) ,
  hasicon(RecordRegion, RecordRegionIcon),
    hasmin(SetOfRegions, ReMin), RegionsMin >= ReMin,
    hasmax(SetOfRegions, ReMax), ReMax >= RegionsMax,
  has(RecordRegion, RecordInfo) ,
  hasicon(RecordInfo, RecordInfoIcon),
    owns(RecordInfo, AttributeChart) ,
    isof(AttributeChart, typeofdatapicture) ,
    owns(RecordInfo, AttributeNotes) ,
    isof(AttributeNotes, typeofdatatext) ,
  owns(RecordRegion, AttributeName) ,
  isof(AttributeName, typeofdatastring) ,
  contains(RecordRegion, SetOfProvince) ,
    collects(SetOfProvince, RecordProvincia) ,
    hasicon(RecordProvincia, RecordProvinciaIcon),
      hasmin(SetOfProvince, PrMin), ProvinceMin >= PrMin,
      hasmax(SetOfProvince, PrMax), PrMax >= ProvinceMax,
    owns(RecordProvincia, AttributeCitta),
    isof(AttributeCitta, typeofdatastring),
    contains(RecordProvincia, SetOfMonuments) ,
      collects(SetOfMonuments, RecordMonument) ,
      hasicon(RecordMonument, RecordMonumentIcon),
        hasmin(SetOfMonuments, MoMin), MonumentsMin >= MoMin,
        hasmax(SetOfMonuments, MoMax), MoMax >= MonumentsMax,
      owns(RecordMonument, AttributePhoto) ,
      isof(AttributePhoto, typeofdatapicture) ,
      owns(RecordMonument, AttributeNameMon) ,
      isof(AttributeNameMon, typeofdatastring) ,
    contains(RecordProvincia, SetOfTowns) ,
      collects(SetOfTowns, RecordTown) ,
      hasicon(RecordTown, RecordTownsIcon),
        hasmin(SetOfTowns, ToMin), TownsMin >= ToMin,
        hasmax(SetOfTowns, ToMax), ToMax >= TownsMax,
      owns(RecordTown, AttributeNameTown) ,
      isof(AttributeNameTown, typeofdatastring) ,
      contains(RecordTown, SetOfLocMons) ,
        collects(SetOfLocMons, RecordLocMon) ,
        hasicon(RecordLocMon, RecordLocMonIcon),
          hasmin(SetOfLocMons, LMMin), LocMonsMin >= LMMin,
          hasmax(SetOfLocMons, LMMax), LMMax >= LocMonsMax,
        owns(RecordLocMon, AttributeNLocMon) ,
        isof(AttributeNLocMon, typeofdatastring),
```

```
                    /*    File Output  */

tell(mdtitaly),
write('THE RECORD ITALY HAS BEEN MAPPED WITH          : '),
write(RecordItaly),nl,
write('ITS ICON IS                                    : '),
write(RecordItalyIcon),nl,nl,
write('THE SET OF REGIONS HAS BEEN MAPPED WITH        : '),
write(SetOfRegions),nl,
write('   MIN WORLD : '),write(ReMin),
write('   MIN QUERY : '),write(RegionsMin),nl,
write('   MAX WORLD : '),write(ReMax),
write('   MAX QUERY : '),write(RegionsMax),nl,nl,
write('THE RECORD REGION HAS BEEN MAPPED WITH         : '),
write(RecordRegion),nl,
write('ITS ICON IS                                    : '),
write(RecordRegionIcon),nl,nl,
write('THE RECORD INFO HAS BEEN MAPPED WITH           : '),
write(RecordInfo),nl,
write('ITS ICON IS                                    : '),
write(RecordInfoIcon),nl,nl,
write('THE ATTRIBUTE CHART HAS BEEN MAPPED WITH       : '),
write(AttributeChart),nl,
write('THE ATTRIBUTE NOTES HAS BEEN MAPPED WITH       : '),
write(AttributeNotes),nl,
write('THE ATTRIBUTE NAME HAS BEEN MAPPED WITH        : '),
write(AttributeName),nl,
write('THE SET OF PROVINCE HAS BEEN MAPPED WITH       : '),
write(SetOfProvince),nl,
write('   MIN WORLD : '),write(PrMin),
write('   MIN QUERY : '),write(ProvinceMin),nl,
write('   MAX WORLD : '),write(PrMax),
write('   MAX QUERY : '),write(ProvinceMax),nl,nl,
write('THE RECORD PROVINCIA HAS BEEN MAPPED WITH      : '),
write(RecordProvincia),nl,
write('ITS ICON IS                                    : '),
write(RecordProvinciaIcon),nl,nl,
write('THE SET OF MONUMENTS HAS BEEN MAPPED WITH      : '),
write(SetOfMonuments),nl,
write('   MIN WORLD : '),write(MoMin),
write('   MIN QUERY : '),write(MonumentsMin),nl,
write('   MAX WORLD : '),write(MoMax),
write('   MAX QUERY : '),write(MonumentsMax),nl,nl,
write('THE RECORD MONUMENT HAS BEEN MAPPED WITH       : '),
write(RecordMonument),nl,
write('ITS ICON IS                                    : '),
write(RecordMonumentIcon),nl,nl,
write('THE ATTRIBUTE PHOTO HAS BEEN MAPPED WITH       : '),
write(AttributePhoto),nl,
write('THE ATTRIBUTE NAMEMON HAS BEEN MAPPED WITH     : '),
write(AttributeNameMon),nl,
write('THE SET OF TOWNS HAS BEEN MAPPED WITH          : '),
write(SetOfTowns),nl,
write('   MIN WORLD : '),write(ToMin),
write('   MIN QUERY : '),write(TownsMin),nl,
write('   MAX WORLD : '),write(ToMax),
write('   MAX QUERY : '),write(TownsMax),nl,nl,
write('THE RECORD TOWNS HAS BEEN MAPPED WITH          : '),
write(RecordTown),nl,
write('ITS ICON IS                                    : '),
write(RecordTownsIcon),nl,nl,
write('THE ATTRIBUTE NAME TOWN HAS BEEN MAPPED WITH   : '),
write(AttributeNameTown),nl,
write('THE SET OF LOC MONS HAS BEEN MAPPED WITH       : '),
write(SetOfLocMons),nl,
write('   MIN WORLD : '),write(LMMin),
write('   MIN QUERY : '),write(LocMonsMin),nl,
write('   MAX WORLD : '),write(LMMax),
write('   MAX QUERY : '),write(LocMonsMax),nl,nl,
write('THE RECORD LOC MON HAS BEEN MAPPED WITH        : '),
write(RecordLocMon),nl,
write('ITS ICON IS                                    : '),
write(RecordLocMonIcon),nl,nl,
write('THE ATTRIBUTE NLOC MON HAS BEEN MAPPED WITH    : '),
write(AttributeNLocMon),nl,nl,

write('#####################################################'),
nl,nl,fail.
```

Figure B.2 - Query list for query "Italy"

The first scene of the virtual world “Railway”, contains two nested classifiers (see figure B.3). First, you may choose a train by walking on its departure track, which will lead you to a destination. Second, you may decide to enter a specific wagon for a more detailed choice of the final destination. Every track contains a display divided into three screens, each of them supporting a different kind of data. The destination station also contains two classifiers but not nested this time. You may indeed take another train or enter the underground for reaching a near location.

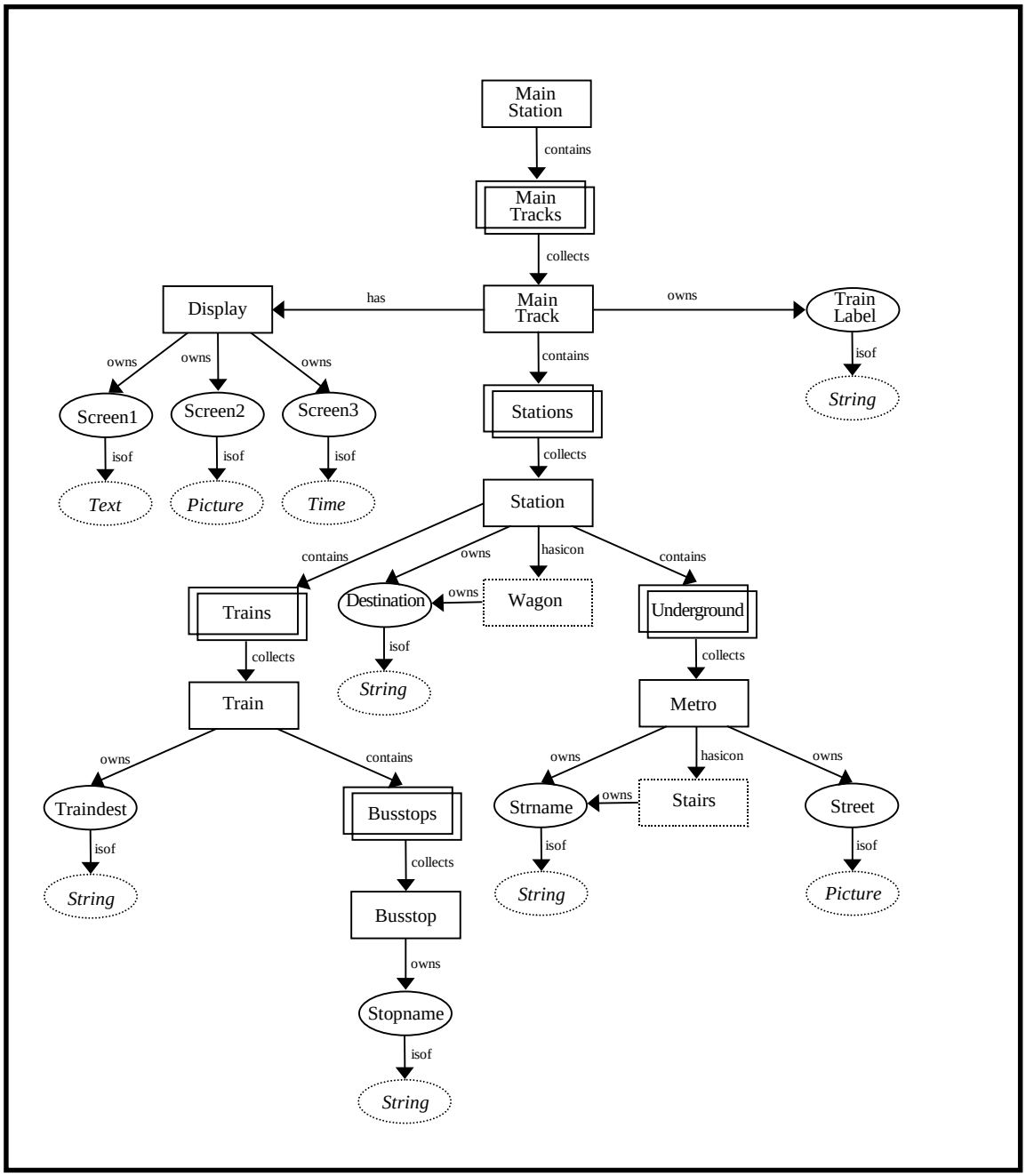


Figure B.3 - Virtual world "Railway"

The table on figure B.3b explains the meaning of the objects of the virtual world "Railway". Use it for a better understanding of figure B.3.

VW OBJECT	NODE TYPE	MEANING
Main Station	Aggregator	Entrance hall of the station
Main Tracks	Classifier	Station passage leading to the departure tracks
Main Track	Aggregator	Single departure track
Display	Aggregator	Multifunction display terminal
Screen1	Accessory	Screen of the display containing some text
Screen2	Accessory	Screen of the display containing a picture
Screen3	Accessory	Screen of the display containing a time value
Train Label	Accessory	Label identifying the train's main destination
Stations	Classifier	Train with different wagons
Wagon	Aggrsymbol	Wagon entrance
Station	Aggregator	A smaller station
Destination	Accessory	Label identifying the name of the station
Underground	Classifier	Passage leading to underground trains
Stairs	Aggrsymbol	Stairs leading to a specific underground train
Metro	Aggregator	Underground train
Strname	Accessory	Label with name of a street
Street	Accessory	Image of the reached place
Trains	Classifier	Passage leading to the departure tracks
Train	Aggregator	Single departure track
Traindest	Accessory	Label identifying the destination of a train
Busstops	Classifier	Timetable for bus departures
Busstop	Aggregator	Entry of the timetable
Stopname	Accessory	Label identifying the destination of a bus

Figure B.3b - Meaning of the objects contained in the graph of figure B.3.

Virgilio will be more efficient and useful when a large number of virtual worlds are available. There should be no limit set to the fantasy of virtual-world designers. An airport, a trade fair or a ship would also be suitable solutions for displaying large amounts of data. It could be even possible to think of combinations among different worlds, as long as the metaphor effect is considered.

The Prolog list for the virtual world "railway" is shown in Figure B.4.

```
                /* VIRTUAL WORLD RAILWAY */


contains(aggregatorMainStation, classifierMainTracks).
contains(aggregatorMainTrack, classifierStations).
contains(aggregatorStation, classifierUnderground).
contains(aggregatorStation, classifierTrains).
contains(aggregatorTrain, classifierBusStops).

collects(classifierMainTracks, aggregatorMainTrack).
collects(classifierStations, aggregatorStation).
collects(classifierUnderground, aggregatorMetro).
collects(classifierTrains, aggregatorTrain).
collects(classifierBusStops, aggregatorBusStop).

has(aggregatorMainTrack, aggregatorDisplay).

owns(aggregatorDisplay, accessoryScreenOne).
owns(aggregatorDisplay, accessoryScreenTwo).
owns(aggregatorDisplay, accessoryScreenThree).
owns(aggregatorMainTrack, accessoryTrainLabel).
owns(aggregatorStation, accessoryDestination).
owns(aggregatorMetro, accessoryStreet).
owns(aggregatorMetro, accessoryStrName).
owns(aggregatorTrain, accessoryTrainDest).
owns(aggregatorBusStop, accessoryStopName).
owns(aggrsymbolWagon, accessoryDestination).
owns(aggrsymbolStairs, accessoryStrName).

isof(accessoryScreenOne, typeofdatatext).
isof(accessoryScreenTwo, typeofdatapicture).
isof(accessoryScreenThree, typeofdatatime).
isof(accessoryTrainLabel, typeofdatastring).
isof(accessoryDestination, typeofdatastring).
isof(accessoryStreet, typeofdatapicture).
isof(accessoryStrName, typeofdatastring).
isof(accessoryTrainDest, typeofdatastring).
isof(accessoryStopName, typeofdatastring).

hasmin(classifierMainTracks, 2).
hasmin(classifierStations, 1).
hasmin(classifierUnderground, 1).
hasmin(classifierTrains, 1).
hasmin(classifierBusStops, 1).

hasmax(classifierMainTracks, 25).
hasmax(classifierStations, 20).
hasmax(classifierUnderground, 20).
hasmax(classifierTrains, 15).
hasmax(classifierBusStops, 30).

hasicon(aggregatorStation, aggrsymbolWagon).
hasicon(aggregatorMetro, aggrsymbolStairs).
hasicon(aggregatorMainTrack, nil).
hasicon(aggregatorTrain, nil).
hasicon(aggregatorMainStation, nil).
hasicon(aggregatorDisplay, nil).
hasicon(aggregatorBusStop, nil).
```

Figure B.4 - Virtual world "Railway" list

The command file related to the mapping process for the query "Italy" (figure B.5) contains the loading of both virtual worlds and, of course, the loading of the query list. The system administrator or other tools may have selected to add the "Railway" virtual world after considering the geographical nature of the query. In this way a more efficient metaphor may be generated.

The MDT finds out three successful mappings (shown on the next pages). The first two pertain to the virtual world "Building". They differ as regards the chosen objects in a room. In the first case the classifier chestOfDrawers has been chosen twice, and in the second case a classifier album replaces one chest of drawers. The third mapping contains the objects of the virtual world "Railway".

```
/*  COMMAND FILE FOR THE MAPPING OF QUERY 'ITALY'   */

      /*  LOADING OF PREFERENCES   */

            [multi].

      /*  LOADING OF VIRTUAL WORLDS   */

            [building].
            [railway].

      /*   LOADING OF THE QUERY   */

            [italy].

      /*   GOAL TO REACH   */

            main.
```

Figure B.5 - Command File for the mapping process of query 'Italy'

The first mapping result regards the virtual world "Building". Its entrance hall leads to an elevator which gives the possibility to access different floors of the building (by pushing a button on the button panel). Each floor is mapped to an Italian region. Once arrived on a selected floor the scene visualises an information board and a corridor of rooms. The board consists of two displays. The former shows a geographical map of the region belonging to that floor. The latter contains some historical notes about that region. Each room represents a province and contains two chests of drawers displaying the requested data. The first chest visualises the names and photos of the main tourist attractions of the province on the front panel of each drawer. The second chest displays on its panel only the name of a town with less than 30.000 inhabitants. By opening a drawer of this chest it is possible to see some folders which contain the names of the main tourist attractions of the corresponding town.

```
               /*   MAPPING RESULTS QUERY "ITALY"   */

THE RECORD ITALY HAS BEEN MAPPED WITH        : aggregatorelevator
ITS ICON IS                                  : nil

THE SET OF REGIONS HAS BEEN MAPPED WITH      : classifierbuttontable
   MIN WORLD : 2   MIN QUERY : 20
   MAX WORLD : 20   MAX QUERY : 20
THE RECORD REGION HAS BEEN MAPPED WITH       : aggregatorfloor
ITS ICON IS                                  : aggrsymbolbutton

THE RECORD INFO HAS BEEN MAPPED WITH         : aggregatorcompboard
ITS ICON IS                                  : nil

THE ATTRIBUTE CHART HAS BEEN MAPPED WITH     : accessorypicture
THE ATTRIBUTE NOTES HAS BEEN MAPPED WITH     : accessoryboard
THE ATTRIBUTE NAME HAS BEEN MAPPED WITH      : accessoryfname
THE SET OF PROVINCE HAS BEEN MAPPED WITH     : classifiercorridor
   MIN WORLD : 1   MIN QUERY : 1
   MAX WORLD : 30   MAX QUERY : 15

THE RECORD PROVINCIA HAS BEEN MAPPED WITH    : aggregatorroom
ITS ICON IS                                  : aggrsymboldoor

THE ATTRIBUTE CITY HAS BEEN MAPPED WITH      : accessorydlabel
THE SET OF MONUMENTS HAS BEEN MAPPED WITH    : classifierchestdrawers
   MIN WORLD : 0   MIN QUERY : 1
   MAX WORLD : 20   MAX QUERY : 20

THE RECORD MONUMENT HAS BEEN MAPPED WITH     : aggregatordrawer
ITS ICON IS                                  : aggrsymboldrawerfront

THE ATTRIBUTE PHOTO HAS BEEN MAPPED WITH     : accessorydrawerpicture
THE ATTRIBUTE NAMEMON HAS BEEN MAPPED WITH   : accessorydrawerlabel
THE SET OF TOWNS HAS BEEN MAPPED WITH        : classifierchestdrawers
   MIN WORLD : 0   MIN QUERY : 1
   MAX WORLD : 20   MAX QUERY : 8

THE RECORD TOWNS HAS BEEN MAPPED WITH        : aggregatordrawer
ITS ICON IS                                  : aggrsymboldrawerfront

THE ATTRIBUTE NAME TOWN HAS BEEN MAPPED WITH : accessorydrawerlabel
THE SET OF LOC MONS HAS BEEN MAPPED WITH     : classifierfolders
   MIN WORLD : 1   MIN QUERY : 1
   MAX WORLD : 10   MAX QUERY : 10

THE RECORD LOC MON HAS BEEN MAPPED WITH      : aggregatorfolder
ITS ICON IS                                  : nil

THE ATTRIBUTE NLOC MON HAS BEEN MAPPED WITH  : accessoryfoname

########################################################################
```

Figure B.6 - 1. Mapping result for query "Italy" with virtual world "Building" .

This first mapping would probably be discarded and therefore not shown to the final users for two reasons. First of all, it contains a repetition of two objects (the chest of drawers in a room) which may cause perplexity among the users. Furthermore, the drawers representing the tourist attractions of the regions are empty, and may produce undesired metaphoric effects.

Look at the metaphor graph for this mapping on figure B.7.

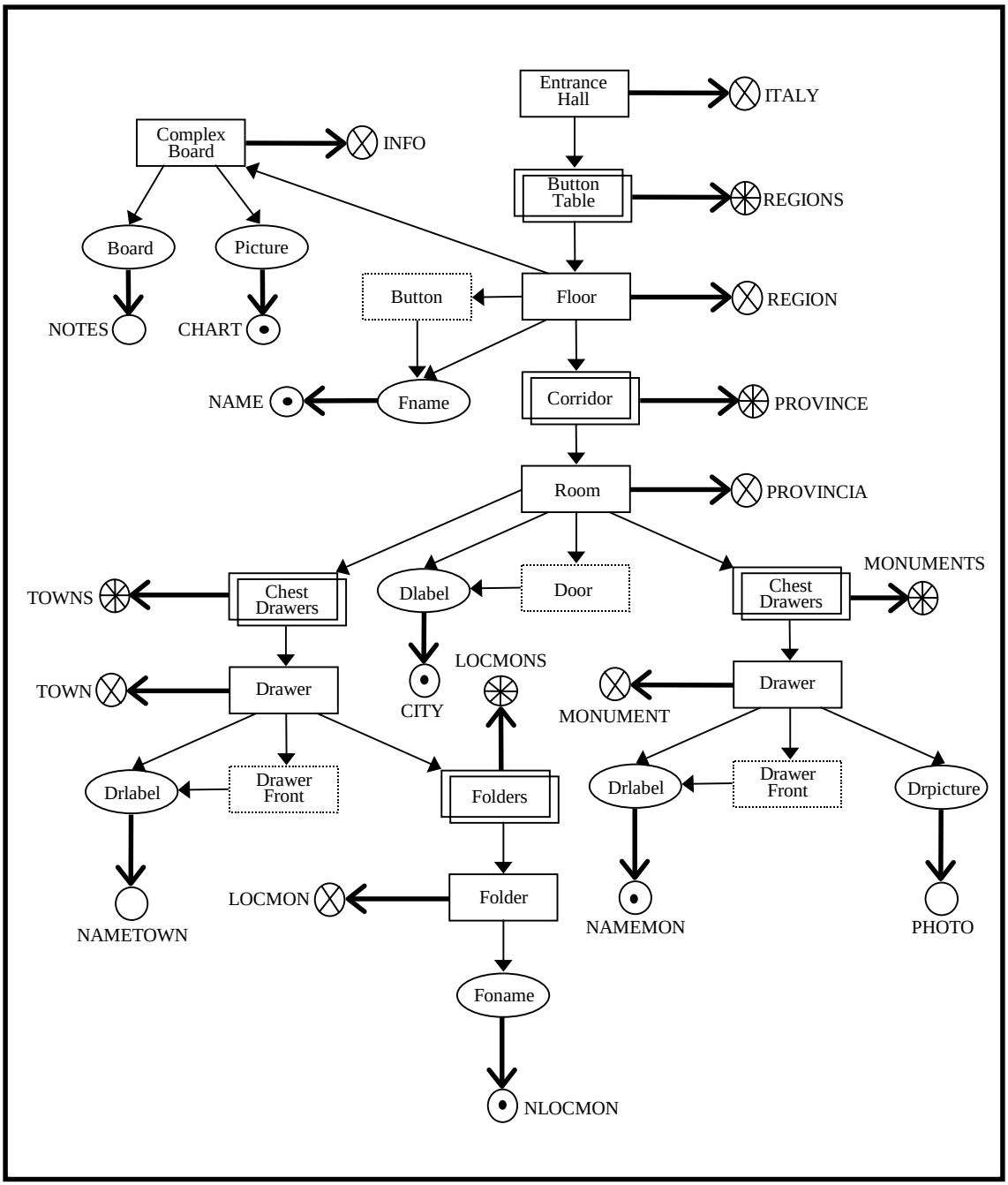


Figure B.7 - Metaphor graph for the first mapping result of query "Italy"

The second mapping result differs from the first one only as regards the objects in a room. One chest of drawers has been replaced by a classifier album. It visualises the names and photos of the main tourist attractions of the province related to the selected room. Figure B.8 shows the second mapping results also stored into the file "mdtitaly".

```
                  /*   MAPPING RESULTS QUERY "ITALY"   */

THE RECORD ITALY HAS BEEN MAPPED WITH        : aggregatorelevator
ITS ICON IS                                  : nil

THE SET OF REGIONS HAS BEEN MAPPED WITH      : classifierbuttontable
   MIN WORLD : 2   MIN QUERY : 20
   MAX WORLD : 20   MAX QUERY : 20

THE RECORD REGION HAS BEEN MAPPED WITH       : aggregatorfloor
ITS ICON IS                                  : aggrsymbolbutton

THE RECORD INFO HAS BEEN MAPPED WITH         : aggregatorcompboard
ITS ICON IS                                  : nil

THE ATTRIBUTE CHART HAS BEEN MAPPED WITH     : accessorypicture
THE ATTRIBUTE NOTES HAS BEEN MAPPED WITH     : accessoryboard
THE ATTRIBUTE NAME HAS BEEN MAPPED WITH      : accessoryfname
THE SET OF PROVINCE HAS BEEN MAPPED WITH     : classifiercorridor
   MIN WORLD : 1   MIN QUERY : 1
   MAX WORLD : 30   MAX QUERY : 15

THE RECORD PROVINCIA HAS BEEN MAPPED WITH    : aggregatorroom
ITS ICON IS                                  : aggrsymboldoor

THE ATTRIBUTE CITY HAS BEEN MAPPED WITH      : accessorydlabel
THE SET OF MONUMENTS HAS BEEN MAPPED WITH    : classifieralbum
   MIN WORLD : 1   MIN QUERY : 1
   MAX WORLD : 30   MAX QUERY : 20

THE RECORD MONUMENT HAS BEEN MAPPED WITH     : aggregatorpage
ITS ICON IS                                  : aggrsymbolindex

THE ATTRIBUTE PHOTO HAS BEEN MAPPED WITH     : accessoryphoto
THE ATTRIBUTE NAMEMON HAS BEEN MAPPED WITH   : accessoryplabel
THE SET OF TOWNS HAS BEEN MAPPED WITH        : classifierchestdrawers
   MIN WORLD : 0   MIN QUERY : 1
   MAX WORLD : 20   MAX QUERY : 8

THE RECORD TOWNS HAS BEEN MAPPED WITH        : aggregatordrawer
ITS ICON IS                                  : aggrsymboldrawerfront

THE ATTRIBUTE NAME TOWN HAS BEEN MAPPED WITH : accessorydrawerlabel
THE SET OF LOC MONS HAS BEEN MAPPED WITH     : classifierfolders
   MIN WORLD : 1   MIN QUERY : 1
   MAX WORLD : 10   MAX QUERY : 10

THE RECORD LOC MON HAS BEEN MAPPED WITH      : aggregatorfolder
ITS ICON IS                                  : nil

THE ATTRIBUTE NLOC MON HAS BEEN MAPPED WITH  : accessoryfoname

######################################################################
```

Figure B.8 - 2. Mapping result for query "Italy" with virtual world "Building".

This mapping result may probably have a better metaphor effect than the precedent, since it contains an object (the album) which seems to be more appropriate for the visualisation of the data type picture.

Figure B.9 shows the metaphor graph for mapping result number two.

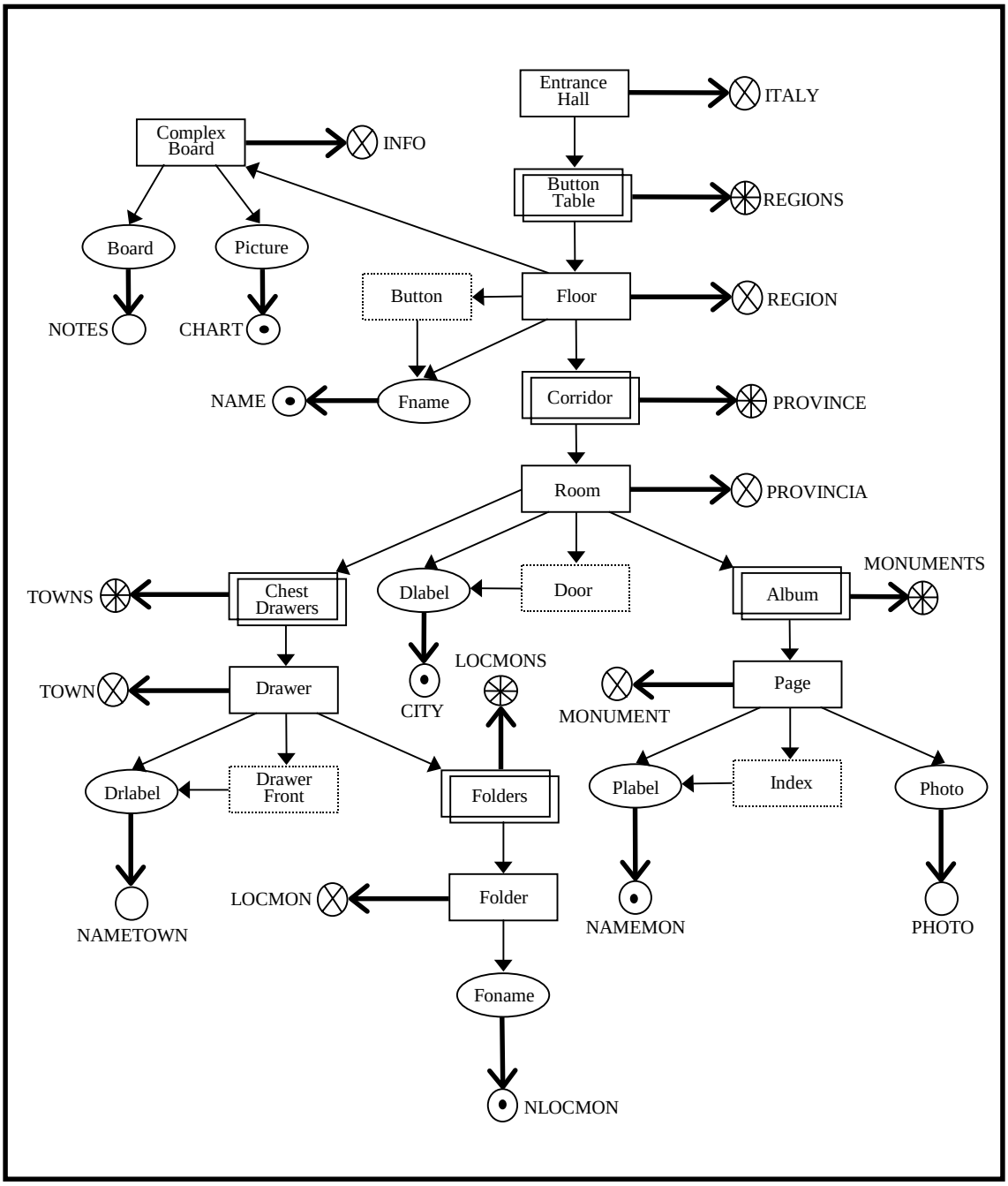


Figure B.9 - Metaphor graph for the second mapping result of query "Italy"

The third and last mapping result concerns the virtual world "Railway" introduced in this section. The first scene represents the main track of the virtual station and allows to access different trains leading to all Italian regions. Once on a departure track you may enter a wagon for the final destination choice, i.e. a province of the selected region. Each track also contains a display with two activated screens. The virtual world foresees three different screens but for the query "Italy" only two are needed, the first one displaying the geographical map and the second one visualising historical notes. Once arrived at the destination station you may decide to take another train for reaching the cities with less than 30000 inhabitants. These trains have only one wagon and the destination station has a simple structure since it allows only to read the bus stop time chart, which shows the names of the tourist attractions for that town. The station of the province gives also the possibility to use an underground system, leading to local monuments and attractions.

```
            /*   MAPPING RESULTS QUERY "ITALY"   */

THE RECORD ITALY HAS BEEN MAPPED WITH        : aggregatorMainStation
ITS ICON IS                                  : nil

THE SET OF REGIONS HAS BEEN MAPPED WITH      : classifierMainTracks
   MIN WORLD : 2   MIN QUERY : 20
   MAX WORLD : 25   MAX QUERY : 20

THE RECORD REGION HAS BEEN MAPPED WITH       : aggregatorMainTrack
ITS ICON IS                                  : nil

THE RECORD INFO HAS BEEN MAPPED WITH         : aggregatorDisplay
ITS ICON IS                                  : nil

THE ATTRIBUTE CHART HAS BEEN MAPPED WITH     : accessoryScreenTwo
THE ATTRIBUTE NOTES HAS BEEN MAPPED WITH     : accessoryScreenOne
THE ATTRIBUTE NAME HAS BEEN MAPPED WITH      : accessoryTrainLabel
THE SET OF PROVINCE HAS BEEN MAPPED WITH     : classifierStations
   MIN WORLD : 1   MIN QUERY : 1
   MAX WORLD : 20   MAX QUERY : 15

THE RECORD PROVINCIA HAS BEEN MAPPED WITH    : aggregatorStation
ITS ICON IS                                  : aggrsymbolWagon

THE ATTRIBUTE CITY HAS BEEN MAPPED WITH      : accessorydestination
THE SET OF MONUMENTS HAS BEEN MAPPED WITH    : classifierUnderground
   MIN WORLD : 1   MIN QUERY : 1
   MAX WORLD : 20   MAX QUERY : 20

THE RECORD MONUMENT HAS BEEN MAPPED WITH     : aggregatorMetro
ITS ICON IS                                  : aggrsymbolStairs

THE ATTRIBUTE PHOTO HAS BEEN MAPPED WITH     : accessoryStreet
THE ATTRIBUTE NAMEMON HAS BEEN MAPPED WITH   : accessoryStreetName
THE SET OF TOWNS HAS BEEN MAPPED WITH        : classifierTrains
   MIN WORLD : 1   MIN QUERY : 1
   MAX WORLD : 15   MAX QUERY : 8

THE RECORD TOWNS HAS BEEN MAPPED WITH        : aggregatorTrain
ITS ICON IS                                  : nil

THE ATTRIBUTE NAME TOWN HAS BEEN MAPPED WITH : accessoryTrainDest
THE SET OF LOC MONS HAS BEEN MAPPED WITH     : classifierBusStops
   MIN WORLD : 1   MIN QUERY : 1
   MAX WORLD : 30   MAX QUERY : 10

THE RECORD LOC MON HAS BEEN MAPPED WITH      : aggregatorBusStop
ITS ICON IS                                  : nil

THE ATTRIBUTE NLOC MON HAS BEEN MAPPED WITH  : accessoryStopName

########################################################################
```

Figure B.10 - 3. Mapping result for query "Italy"
with virtual world "Railway".

Mapping result number three could be preferred to number two because of the geographical nature of the virtual world "Railway". Anyway, the final decision could also be taken by the users, according to their needs or preferences.
The metaphor graph for the third mapping is depicted in figure B.11.

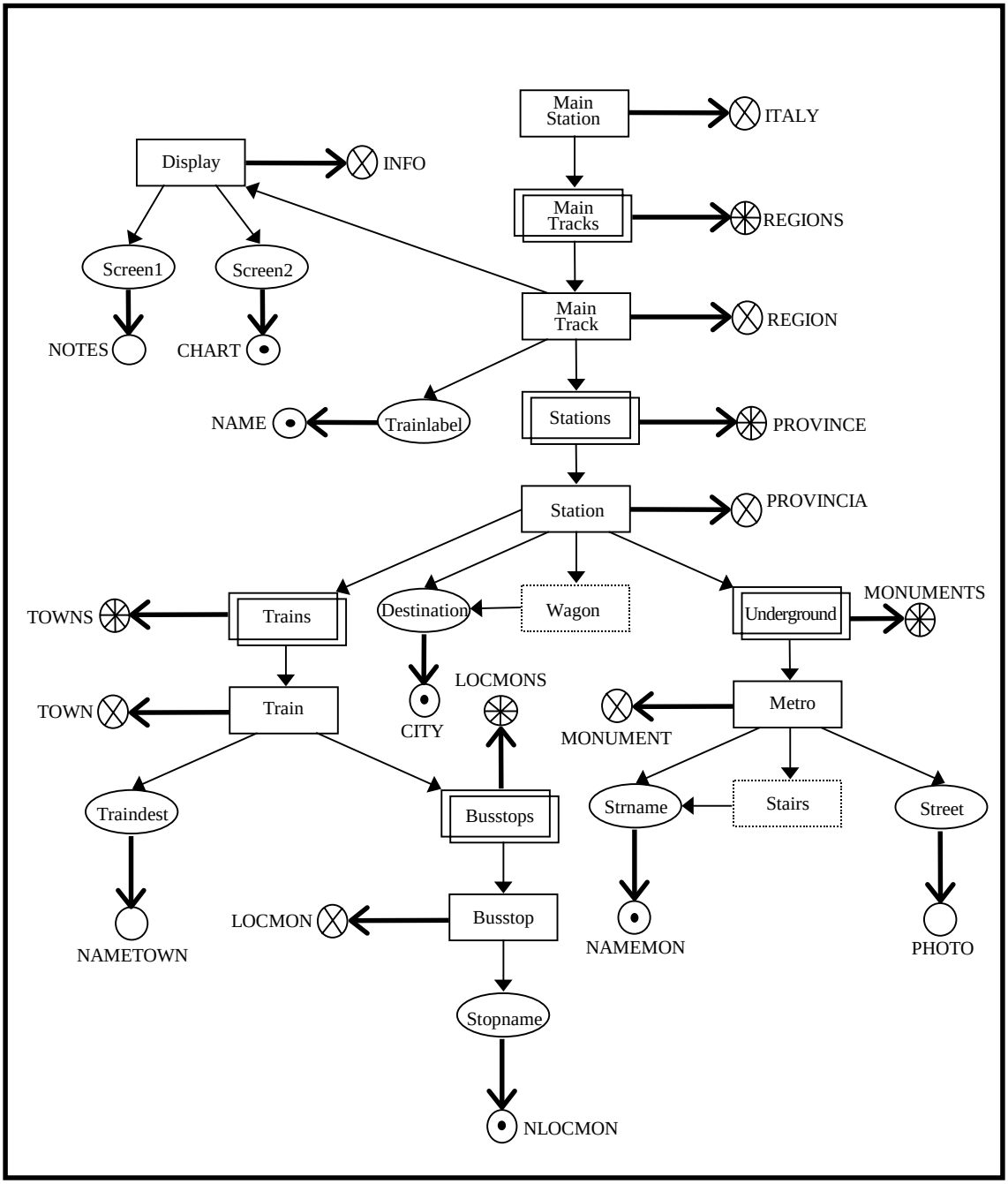


Figure B.11 - Metaphor graph for the third mapping result of query "Italy"