A Consistent and Efficient Graphical User Interface Design and Querying Organelle Genome "GUEDOS"

Hassan BADIR hbadir@gmail.com
Labtic, National School of applied sciences
University Abdelmalek essaadi
90020, Tangier, Morocco

Rachida FISSOUNE fissoune@gmail.com
National School of applied sciences
University Abdelmalek essaadi
Tetouan, Morocco

Amjad RATTROUT ramjad@gmail.com
University Claude Bernard
Lyon, France

Abstract

We propose a software layer called GUEDOS-DB upon Object-Relational Database Management System ORDMS. In this work we apply it in Molecular Biology, more precisely Organelle complete genome. We aim to offer biologists the possibility to access in a unified way information spread among heterogeneous genome databanks. In this paper, the goal is firstly, to provide a visual schema graph through a number of illustrative examples. The adopted, human-computer interaction technique in this visual designing and querying makes very easy for biologists to formulate database queries compared with linear textual query representation.

Keywords: Graphical User Interface, Complex Object, Bioinformatics, Gene, Genome, UML-BD

1. INTRODUCTION

Many biologists use freely and extensively a large number of databanks collecting considerable amount of information. Data from sequence databanks are daily submitted worldwide, usually by electronic mail, then manually checked and stored – generally-under a RDBMS. They are then broad casted to the main servers of the community – daily by Internet or quarterly by CDROM-under a flat file format (ASCII files).

There is no agreement on the format and, for the same data; many formats may b available, depending on the databank manager (Europe, US, Japan…). The number and the size of databanks are grouping rapidly (the size of major sequence databanks doubles each year).

Why Organelle Genome? Organelles (mitochondria and chloroplasts) are of interest for several reasons, including their:

- Possible bacterial origins,
- Relationship to the evolution of the nuclear genome,
- Central role in eukaryotic cell energy production
- Utility as population markers.

The most important advantage to genomics offered by organelles is the number of completely sequenced genomes already available and currently being sequenced (e.g. by OGMP: Organelle Genome Mega-sequencing Program). No larger collection of completely sequenced exists. Sequenced organelle genomes are information-rich datasets: firstly most of them belong to a highly analysed set including protein-coding, transfer RNA (tRNA) and ribosomal RNA (rRNA) genes, secondly the relationships between and among genomes can be determined (at both gene and genome levels), making them ideal for comparative genomic studies.

For many years, computer scientists have provided help to biologists for managing these data. As a result, a lot of retrieval systems, like SRS [11] or ACNUC, have been created which are based on indexing flat files. These simple tools obtain a wide success in the biologist community.
Web interfaces have also become very familiar and make the data very easy to access for any biologist. The ability to retrieve data from anywhere on the network has progressively leaded to a new problem: how to interconnect heterogeneous databanks? At present it is difficult for researchers to access all the relevant information associated with an sequence genome. Data are dispersed among a number of sources. In this present disorganized state, organelle genomic data constitute a major underexploited source of information. A wide discussion on this topic has started within the molecular biology community. To integrate such data warehouse is considered as one of the main problems to face up in bioinformatics.
Directly manipulating visual conceptual schemes of complex objects provide users with a clear and powerful mean of interaction. An end user of GUEDOS-DB is able to graphically build a database schema or modify an existing one, to load all the information initially provided by a data bank as GENBANK, to browse through the schema of the database in order to construct the queries about the data and to save them for later use. All of these activities are accomplished by using a unique graphical iconic representation.
As a consequence, different types of users, especially novices, can learn and use the query facilities in a more intuitive way, without necessity to remember the database scheme or the grammar of the query language.
In this paper we describe the visual query interface GUEDOS-DB (DBioMics) developed in the in object-oriented [2] environment for software engineering workshop development. A database schema based on an object-relational [1] data model using SQL3.
The user interface proposed in this paper:

- In descriptive: it provides a concise and complete visualization of the data scheme called Object Semantic Graph;
- Uses the same medium both for the description of the data scheme and for the representation of the formulated query on the semantic graph.
- Is interactive: the formulation of a query is made by simply designating the nodes and arcs of the displayed semantic graph and the syntactic units through the technique of direct manipulation.

The remainder of this paper is organized as follows. We first review the previously proposed biological databases in the next section. In section 3, we review the basic concepts of the object oriented data model. We then describe our proposed graphics user interfaces in section 4. Section 5 describes some graphical queries with a number of illustrative examples. Finally, we conclude in section 6.

2. RELATED WORK

In fact biological interfaces are developed upon one or more database management systems or one or more data file management systems. Database models can be relational, object-Oriented, object-relational or hierarchical. Some ad-hoc solutions have been proposed:

- The **Kabat antibody databank** [4] is one of the first realizations in this area. The system visualizes hierarchical schemes and includes computational tools. It is developed on P/FDM (Functional Data Model), an in-house OODBMS built at Aberdeen. Data are modelized in the FDM, and queries are made through DAPLEX language, which is based on PROLOG. Describing database schemes and querying are quite distinct.
- **GOBASE** [3] is implemented under SYBASE RDMS on a Sun SPARCstation. Custom software has been developed for make easy to populate and maintain the database. It uses the Perl language and Sybase's Open-Client development tools. The query interface is supported through WWW forms using the web/General gateway. The user interface allows the users to navigate in and to retrieve information from GOBASE, and includes the following entities: gene, intron, RNA, protein, organism, sequence, chromosome and signal. The interface also contains hypertext links to specific information in other internet-accessible databases. In future versions, the user interface will be expanded by adding new entities to the database and by offering analytical tools such as subsequence extraction and neighboring searches.

- **AGIS** (*agricultural Genome Information System)* [12] is a World Wide Web product of a cooperative effort between the department of Plant Biology, University of Maryland and College Park in USA. This databank consists of genome information for agricultural organisms. At present, it encompasses mostly crop and livestock or non-commodity animal species. Also included are a number of databases that have related information, e.g. germplasm and plant gene nomenclature data.
- **Genomic databases** can be viewed by ACDB [9] a widely used generic genome interface. It offers a specific visual interface. It is worth noting that this interface supports describing and querying schema but possible queries are too coarse.
- **Docking-D** [3], a prototype for managing ligands (PDB, HPSS…), is implemented with and OODBMS prototype VODAK [8]. According to its authors, the VODAK data model provides all standard features of object-oriented data model; the system includes SQL-Like query language with optimization and multi-user access modules. VODAK was initially created in response to the lack of a declarative query language in many object-oriented database systems like ObjectStore. It is still under development.

3. GUEDOS DESCRIBING

3.1 Architecture

3.1.1 Description

GUEDOS prototype is based on a graphical approach used to represent graphically database schemas. In addition, schema conception is realized by systemic and direct graphs manipulations by a set of available graphic operators. Also, GUEDOS is characterized by several criteria:

- **Flexible**: flexibility implies that the used graphical language should be adapted to possible uses.
- **Incremental**: practical interest is situated in the fact that, during schema conception phase, user can construct schema incrementally by defining more constraints.
- **Uniform**: uniformity imposes identical interaction modes for the different functionalities enhancing thus the tool ergonomic.

Figure 1 describes GUEDOS general architecture. It's composed mainly by two components: a graphical interface and an automatic transformation module.

GUEDOS Prototype was developed by java. It represents a semantic suite of LASCI-Complex toot developed earlier by in a thesis framework. This tool allows an automatic visualization of database structure, allowing thus an automatic transformation using algorithms developed by our team.

3.1.2 Functions

GUEDOS is a platform for editing and designing object and object relational database schemas. The systemic aspect in GUEDOS is the key solution for a guided construction of such a schema, because his objective is to facilitate requirements specifications of skilled designer or not, and to simplify him the design task. Its functions are to allow:

- To users to express their requirements by the means of one or several models like: universal relation with inclusion (URI), object attributes forest and UML DB-stereotyped classes diagram. The tool deals with these models by the means of a logical and syntax apprehension of those. User describe his requirements, and affine his constraints to obtain an initial conceptual schema. User can also normalize and transform a representation under the form of relations related by inclusions dependences into a normalized semantic graph. This normalization could be partial, leaving unchanged certain complex structures of fixed objects by user,
- To swap from one model -among those cited above- to another using transformation algorithms [6] and to generate SQL3 description or an XML schema,
- To personalize the obtained conceptual schema with respect to the foreseen processing, by introducing access methods, even denormalizing them.
- And to integrate many conceptual schemas into one without losing any information.

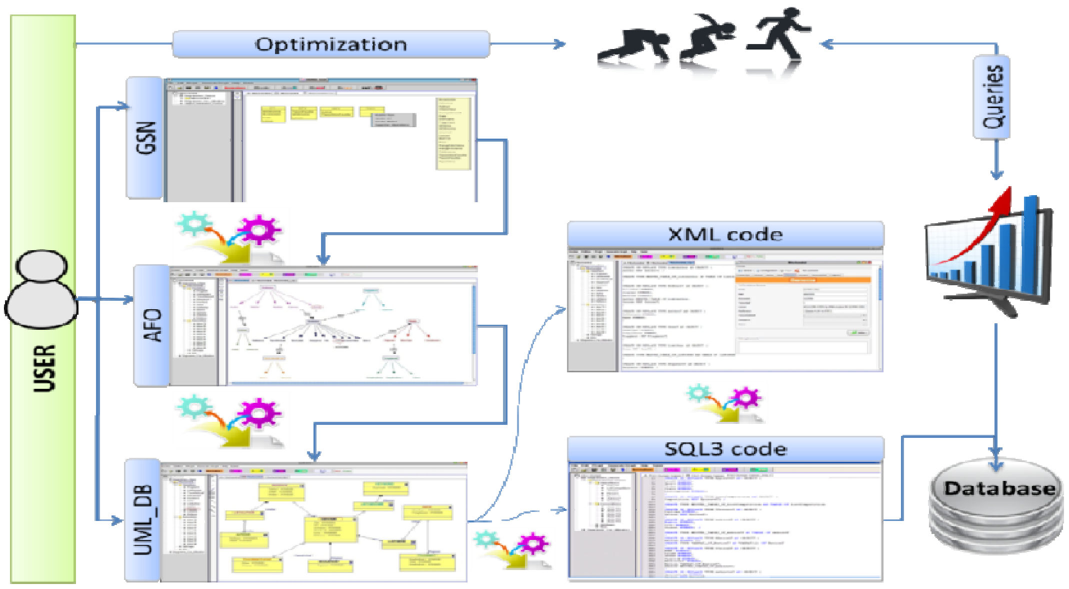


FIGURE 1: Architecture of GUEDOS

GUEDOS interface could be decomposed into three different screens, according to user need. In the first case, user looks to edit a set of attributes and functional dependencies to produce systemically NSG, using normalization algorithm; secondly, user constructs an OAF manually by specifying its constraints or automatically from the previously obtained NSG relying on a transformation algorithm developed in [6]. In the third case, user conceives a database stereotyped UML class diagram from an OAF previously obtained using transformation algorithm in [7]. GUEDOS has two other forms allowing to display SQL3 description and XML schema generated from classes diagram. Concretely, GUEDOS avoids waste of time generally noticed in redoing repetitive tasks, and allows then a more coherent information processing. User beneficing of supervision and control capacities on global tasks he accomplishes. It allows, more generally, a better comprehension of user conceptual framework.

3.2 Manipulation and Importing

The first part of this biological application has consisted in defining a data schema in the Graphical Object Data Model (GODM) [13] [6].

Figure 2 shows mitochondrial sequences graph. This schema respects the ontology and allows the indexation to the largest quantity of knowledge. The design of a GODM schema is not our object in this paper [6][14].

GUEDOS includes a schema editor, allowing designers to build such a GODM schema by picking graphical symbols from palette and positioning them into the workspace provided in an ad-hoc window. An analyst is able to work simultaneously on several schemas by using graphical window for each. Standard editing operations are available through pull-down menus.

The mitochondrial schema [10] represents seven fundamental classes corresponding to the types Genome, Gene, Fragment, *NotLeafTaxon*, *Reference*, *KeyWord* and *Author*. These are depicted using rectangle nodes, indicating that they correspond to abstract data types in the world. These classes are referencing each other using internal identifiers that are not visible to the user. In contrast with abstract types, attributes may be either atomic or complex.

An atomic attribute is depicted using oval such as *GeneType* of class Gene in Figure 3.

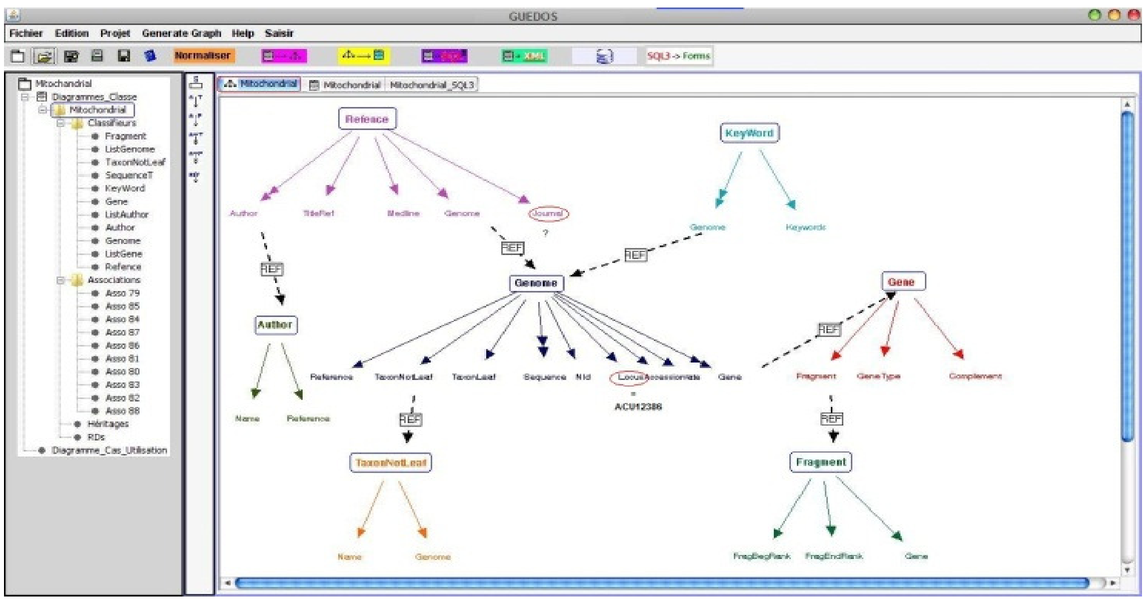


FIGURE 2: schema graph for mitochondrial sequences

A complex attribute is depicted using also oval, but decomposed into a tuple of attributes, which may be also either atomic or complex. Figure 2 also illustrates attribute-domain relationships. For example the *Genome* class has an attribute *NotLeafTaxon* referencing the *NotLeafTaxon* class. Then,

- The *NotLeafTaxon* class is the domain of attribute *NotLeafTaxon* of class **Genome**;
- An object instance of **Genome** has an object instance of *NotLeafTaxon* as the value of *NotLeafTaxon* attribute (single-valued attribute).

Figure 2 shows mitochondrial sequences schema graph. This schema respects the ontology and allows the indexation to the largest quantity of knowledge. In an Object Attributes Forest [7], each attributes forest has a color chosen randomly by GUEDOS or indicated by user. This color will be inherited by UML classes diagram during transformation. A complex class or an attributes tree could have one more keys. For example in figure 2, Genome has as keys *IdGenome* and Locus *TaxonNotLeaf* . Figure 3 represents DB-stereotyped classes diagram with derived keys from an OAF (figure 2) using UML-DB profile [6].

4. GUEDOS MANIPULATION AND QUERIES

There are two ways to extract data from a database. One way is extensional, by navigating through the database at the occurrence level. The other way is intentional, by formulating a query asserting which types in the database schema are relevant to given query and which attribute values and links among objects in order to define which subsets of those populations are relevant. A query also defines which data from the relevant subsets have to be put into the result (typically a projection operation) and how these data have to be structured for the end result (unless the result is by definition a flat or first normal form from a relation). In a classical query language as OQL/SQL, we can find these different specifications expressed as Select-From-Where blocks: the FROM clause restricts these classes to the relevant subsets, finally the Select clause specifies the projected data.

In a visual environment, the definition of the relevant object classes (the FROM clause) is performed by visualizing the database schema on the screen and by clicking on the desired class-nodes to lift them out from the schema into the query sub-schema (alternatively, by clicking on undesired types to reduce them from the schema, which gradually reorients it to the target query sub-schema). The query sub-schema is a sub-graph of the original schema graph.
The definition of the relevant subsets of the classes in the query (the WHERE clause) is expressed as conditions (predicates) upon desired attributes. Only these objects that match the

conditions will contribute to the result. Conditions in a query are divided in two categories: simple conditions which apply to object sets in a single class and composite conditions which apply to object sets in multiple classes.

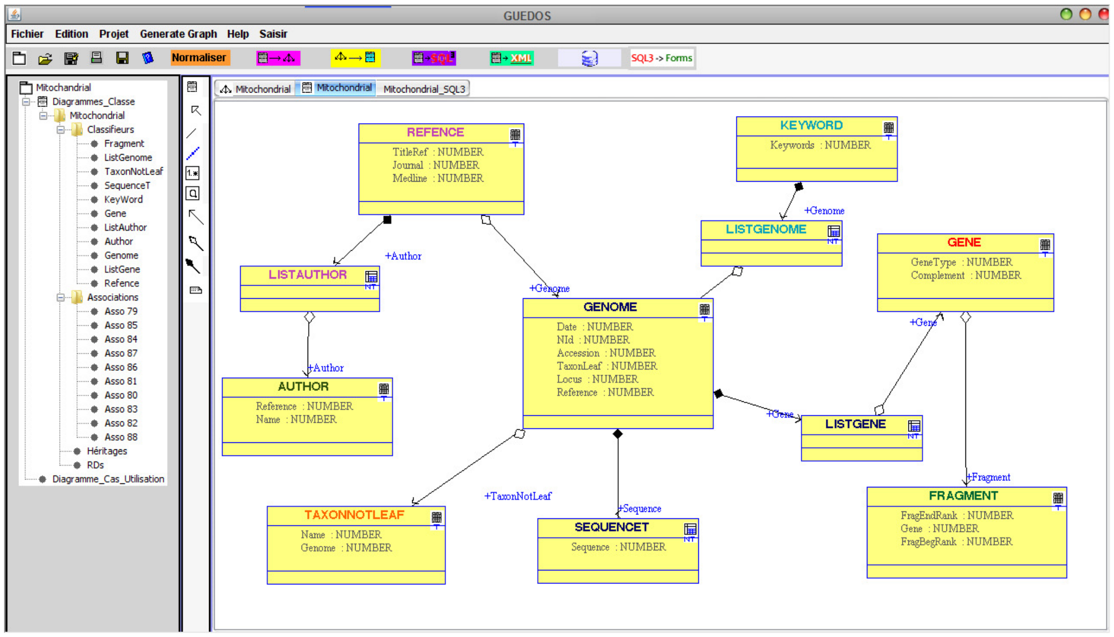


FIGURE 3: schema UML-DB for mitochondrial sequences

The definition of data to be presented to the user and the way by which they have to be presented (the SELECT clause) can be seen as the definition of a new, virtual object type. In the query sub-schema the projected attributes are represented by the symbol "?".

4.1 Complexes Queries in Object-relational Model

Object Relational Queries are the queries, which exploit the basic object relational model thus allowing the user to retrieve data from the new data structures that have been introduced in the model. Object Relational Queries can be broadly classified into: Queries having REF, Queries for Nested Table Structure (Aggregate Queries), Queries using index cluster, and Queries for Inheritance Relationship [16].

REF Queries

REF is incorporated in database by defining one attribute in the table, which holds the REF information of the attribute, which belongs to the other table. REF is used when there is an association relationship between two objects. REF queries are the type of queries, which involve REF either in projection or join or selection. REF is a logical pointer, which creates a link between two tables so that integrity in data can be maintained between the two tables. The attribute which holds the REF value is called as ref attribute (e.g. *Loginstaff_id in Login_t table*) and the attribute to which ref attribute points is called as referred attribute (e.g. *staffid in person_t* table). Ref attributes stores the pointer value of referred attribute as its real data value. Most important thing to keep in mind is that whenever we refer to REF we always give the alias of the table, which holds the referred attribute and not the table name. REF takes as its argument a correlation variable (table alias) for an object table. Generally, REF query consists of Projection, Joins and Selection.

```
SELECT <Projection List>
FROM <table1> <alias1>, <table2> <alias2>, … … …
WHERE <alias2>.<ref attribute> = REF(<alias1>) ;
```

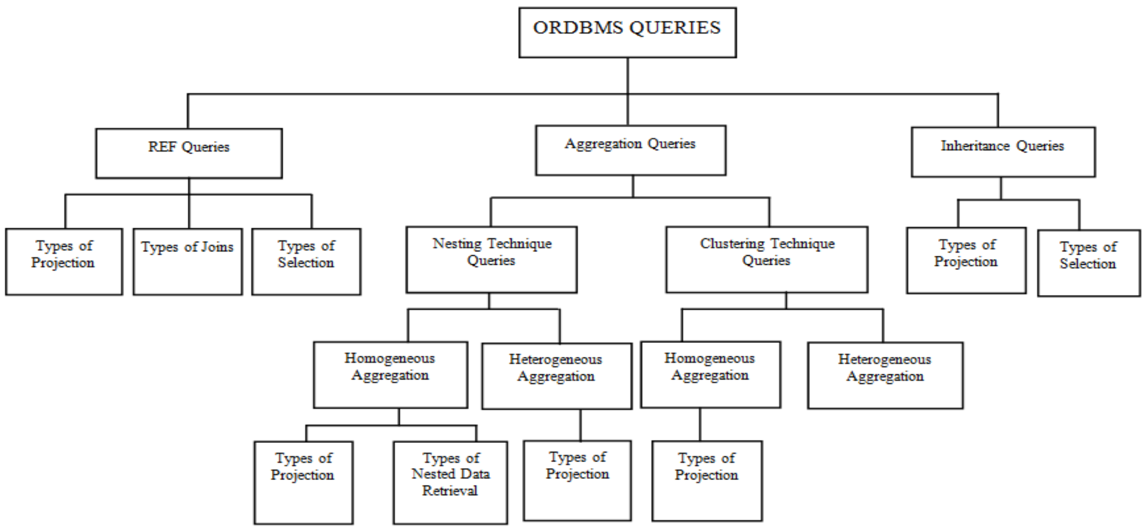


FIGURE 4: General Classification of the ORDBMS Queries

Projection refers to what we to get as a result of the execution of the query. Joins are the links, which are responsible for maintaining integrity in data, which is common to two tables. Selection is the condition based on which we do projection.

Aggregate Queries
Collection types give the flexibility of storing a series of data entries that are jointly associated to a corresponding row in the database. They can be further classified into two: VARRAYS, Nested Tables. We will not be discussing VARRAYS in this section, since at the moment we cannot write SQL statements to access the elements of the VARRAYS. The elements of the VARRAYS can only be accessed through PL/SQL block; hence it is out of the scope of this section.
Nested table is one of the ways for implementing aggregation. Nested table data is stored in a single table, which is then associated with the enclosing table or object type. In nesting technique, the relationship between “part” and “whole” is existence dependent type. If the data for the whole object is removed all of its part objects are removed as well. Nested Table is a user-defined datatype, which is linked to the main table in which it is nested. Generally nesting technique is of two types: Homogeneous and Heterogeneous, depending upon the number of the parts that the main table has.

Aggregation is an abstraction concept for the building composite objects from their component objects. Participating entities have “*Whole-Part*” type of relationship and the part is tightly coupled with whole. Aggregation can be done in the following two ways: Nesting Technique and Clustering Technique. Both the techniques store aggregate data efficiently but the degree of cohesiveness between whole and part data is more in nesting technique than in clustering technique. *Nesting* and *Clustering* technique can further be classified into *Homogeneous* and *Heterogeneous Aggregation* depending upon the number of parts they have,

```
SELECT <Nested table attribute1>,
       <Nested table attribute2>, … … …
FROM THE (SELECT <nested attribute>
          FROM <whole table> <alias1>
          WHERE <primary key condition>);
```

Clustering Technique Aggregation Queries
Clustering gives the flexibility of storing the “whole-part” type of information in one table. This technique is used when there is aggregation (i.e. the whole is composed of parts). This enforces

dependent type of relationship and each (i.e. either whole or part) has unique ID [17]. Clustering can be further classified into homogeneous and heterogeneous clustering aggregation.
Clustering technique implements the participating tables in "Whole–Part" type of relationship. The part information is tightly coupled with the corresponding whole record. For each whole info, we have many corresponding parts and this is achieved by creating cluster on the whole key. Index creation improves the performance of the whole clustering structure. Clustering can also be divided into two types depending upon the number of participating subtypes: Homogeneous Clustering and Heterogeneous Clustering.

```
SELECT CURSOR (SELECT <Projection List>
               FROM <main table name>
               WHERE Join AND[<condition>]),
       CURSOR (SELECT <Projection List>
               FROM <part table name>
               WHERE Join AND [<condition>])
FROM <whole table name> <alias1>,
     <part table name> <alias2>
WHERE <alias1>.<attribute name> = <alias2>.< attribute name >
AND [<condition>];
```

Inheritance Queries

Inheritance is a relationship between entities, which gives the flexibility to have the definition and implementation of one entity to be based on other existing entities. The entities are typically organized into hierarchy consisting of parent and child [17]. Child inherits all the characteristics of its parent and can also add new characteristic of its own. A parent can have many children but each child will have only one parent. There can be multilevel of inheritance (i.e. a parent child can have many other child's as well). Basically in Object Relational Database System, inheritance can be of three types: Union Inheritance, Mutual Exclusion Inheritance and Partition Inheritance. The basic difference between three types of inheritance is in implementation but for querying purposes they are basically the same. In this paper we have only taken Union Inheritance into consideration for writing SQL statements as the only difference between different types of inheritance is the way they are implemented in the database and not in terms of writing SQL. The general syntax for implementing inheritance relationship is as follows.
Generally inheritance queries consist of projection and selection. SQL for inheritance queries is same irrespective of type of inheritance or number of levels in the tree hierarchy unless mentioned specifically. The general syntax for inheritance queries is as follows.

```
SELECT VALUE(<alias>).<supertype attribute name1>,
       VALUE(<alias>).<supertype attribute name2>, … … …
FROM <table name> <alias>;
```

4.2 Query Editor

GUEDOS-ASCK includes an editor for graphical specification of queries, inserts and updates. In this section we present the main features of the editor. The discussion is limited to query formulation. The various steps which the process of query formulation comprises are:

a. Selecting the query sub-schema:

This is the initial step for all visual query languages. The portion relevant to the query is extracted from the database schema is reverse video.

b. Specifying predicates:

Predicates are stated here to be applied to database occurrences, so that only relevant data are selected. Predicates against complex objects may be rather clumsy. For the simplest ones (comparison of a mono-valued attributes with a constant) a graphical counter-part may easily be defined. A simple specification technique is to click on the attributes, select a comparison operator from a menu, and finally type the value or choose one from a list. For complex predicates (involving several quantifiers, for instance), there might be no simple way to express it graphically. Menus are sometimes used for syntactic editing of predicates. In GQL/ER [15], QBE-Like (Kari, 1990) forms are used to specify conditions on the selected nodes. In GUEDOS, textual specification of Boolean or arithmetic expressions has been preferred to graphical representation: There are simpler for complex expressions.

c. Formatting the output:

The selection of projection of projected attributes defines the structure of the resulting entity type.

4.3 Query Examples

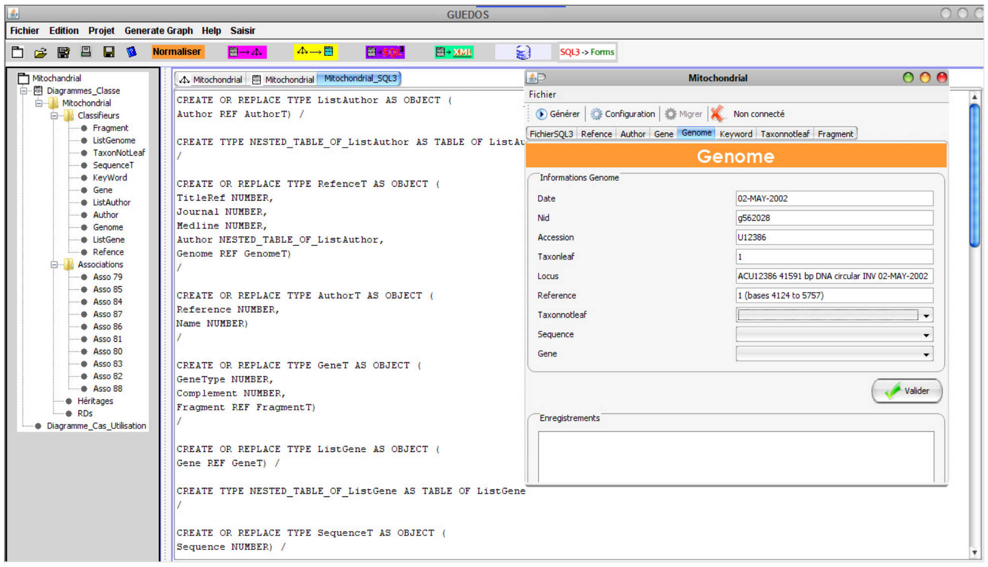
This section illustrates some examples of graphical queries in GUEDOS-Queries. Let us assume that the user wants to formulate a query for the schema show in Figure 1.

For each query we show:

- The **query window**, which displays twos superimposed graphs with different shades:
  - The object-relational schema which is bright and fixed in the background to guide the user,
  - The sub-graph under development, which is reverse video.

  The window has a selection bar which, in addition File and Edit menus, contains: the Query menu, used for choosing the type of query to be made on the query graph
- The **SQL3** code window for displaying the corresponding SQL3 code of the formulated visual query.
- The result window for displaying the result of the executed query.
- Consider the simple query, 'Print the NId and TaxonLeaf of Genome which the Locus is "ACU12386" 'in Figure 4.

  The user selects the Class boxes to which the projection and condition may be specified (Genome Class in this example). The user will proceed in this way.
- Designate the projected attributes ***NId*** and ***TaxonLeaf*** of Genome class by choosing the PROJECT option in the pull-down menu associated to this attributes.
- Establish the selection condition ***Locus*** = "*ACU12386*" of Genome class. The user carries out the following actions:
  - Double-click on the attribute *Locus* of *GENOME* class and the pull-down menu associated to this attribute is displayed,
  - Choose the predicate option in the menu
  - (implicit) choose the comparator "="
  - Implicit choose the option "value to be entered",
  - Enter the value "ACU12386" in the open box.
  - The answer to the request is a temporarily sub-class of the **Genome** class and is composed of appropriate objects. The sub-class name will be either the name of the query or a specific name given by the user. On the query sub-graph, the user selects the SHOW RESULT option in pull-down menu in the class node to visualize the temporarily objects.



```
CREATE OR REPLACE TYPE ListAuthor AS OBJECT (
Author REF AuthorT) /

CREATE TYPE NESTED_TABLE_OF_ListAuthor AS TABLE OF ListAu
/

CREATE OR REPLACE TYPE RefenceT AS OBJECT (
TitleRef NUMBER,
Journal NUMBER,
Medline NUMBER,
Author NESTED_TABLE_OF_ListAuthor,
Genome REF GenomeT)
/

CREATE OR REPLACE TYPE AuthorT AS OBJECT (
Reference NUMBER,
Name NUMBER)
/

CREATE OR REPLACE TYPE GeneT AS OBJECT (
GeneType NUMBER,
Complement NUMBER,
Fragment REF FragmentT)
/

CREATE OR REPLACE TYPE ListGene AS OBJECT (
Gene REF GeneT) /

CREATE TYPE NESTED_TABLE_OF_ListGene AS TABLE OF ListGene
/

CREATE OR REPLACE TYPE SequenceT AS OBJECT (
Sequence NUMBER) /
```

Informations Genome

Date	02-MAY-2002
Nid	g562028
Accession	U12386
Taxonleaf	1
Locus	ACU12386 41591 bp DNA circular INV 02-MAY-2002
Reference	1 (bases 4124 to 5757)
Taxonnotleaf	
Sequence	
Gene	

FIGURE 5: Example of query for mitochondrial sequences

5. CONSLUSION & FUTURE WORK

We have proposed a solution consisting on an integrated environment facilitating cohabitation of several models and techniques to sustain user when designing database schema. GUEDOS is specific to object-relational database schema design.

We have described the environment at whole, focusing at the same time on data static structure and dynamic process modeling. As perspectives, we tend to extend GUEDOS to characterize the obtained conceptual schema with respect to the previous processing, by introducing access methods, even denormalization, and to integrate several schemas conceptual schemas into on schema without lost of information. This work direction leads us to take more attention on optimization and design process using a self-optimization approach based on user preferences by selecting indexing methods, fragmentation and selection of views to materialize.

6. REFERENCES

[1] M. Stonebraker, P. Brown, 1999. Object-Relational DBMSs – Tracking the Next Great Ware, 2nd ed., Morgan Kaufmann, San Fransisco.

[2] Bertino E. and Martino L., 1993. Object Oriented Database Systems; Concepts and Architectures. Addison-Wesley Publishing Company Inc, (1993)

[3] Korab-Laskowska M., Pierre Rioux, Nicolas Brossard, Timothy G. Little-john1, Michael W. Gray2, B. Franz Lang, Gertraud Burger, 2001. The Organelle Genome Database Project (GOBASE). Nucleic Acids Research, volume 26, Issue 01: January 1 (2998) 138-144.

[4] Marie-Paule Lefranc, Véronique Giudicelli, Chantal Busin, Julia Bodmerl, Werner Müller, Ronald Bontrop, Marc Lemaitre, Ansar Malik, Denys Chaume, 1998. IMGT. The international ImMunoGeneTics database, Nucleic Acids Research, Volume 26, Issue 01: Jaunuary 1, 297-303.

[5] Moore, R., Lopes, J., 1999. Paper templates. In TEMPLATE'06, 1st International Conference on Template Production. SciTePress.

[6] Badir, H. and Pichat, E., 2005. An Interactive Tool for creative data modeling, in Database Technology and Applications for International Conference on Information Technology ITCC'05, IEEE, Las Vegas, Nevada

[7] Badir H., Tanacescu A., 2007. An efficient interface to handle complex structure for database design", ICEIS '07, Madeira, Potugal.

[8] Chang W., I.N. Shindyalov, C. Pu and P.E Bourne, 1994. Design and application of PDBlib, a C++ macromolecular class library. Computer Application in Biosciences, 10(6).

[9] Tateno Y., Kaoru Fukami-Kobayashi, Satoru Miyazaki Sugawara and Takashi Gojobori, 1999. DNA Data Bank of Japan at work on genome sequence data. Nucleic Acids Research 6 (19) 16-20.

[10] Lagesen K, et al. RN Ammer, 2007. Consistent and rapid annotation of ribosomal RNA genes. Nucleic Acids Res. 35:3100–3108.

[11] Etzold T. and P. Argos. SRS, 1993. An indexing and retrieval tool for flat file data libraries. Computer Applications in the Biosciences. 9(1) 49-57.

[12] Stephen M. Beckstrom-Sternberg and D. Curtis Jamison, 1999. AGIS: Using the Agricultural Genome Information System, Bioinformatics: Databases and Systems, p 163-174

[13] Kim W., 1989. A model of queries for object-oriented databases. VLDB, pages 423-432.

[14] Kari S. And Rosenthal, A. G-WHIA, 1990. Conceptual Query Language-CQL: a visual user interface to application databases. IOS Press, pages 608-623.

[15] Lecluse, C. Richard, P. And Velez, F. O2, 1988. An Object-Oriented data model. EDBT, pages 556-562.

[16] D. Tania, Rahayu and Srivastava, , 2003, A Taxonomy for Object-Relational Queries, by IRM Press

[17] Loney, K. & Koch, G. (2002). *Oracle 9i: The Complete Reference.* Oracle Press.