# Use of Active Rules in the Schema Evolution Process of an Object Manager System

Ana Maria de C. Moura, Antonio Ricardo N. Guimarães, Ricardo Barros

Instituto Militar de Engenharia - IME

Depto Engenharia de Sistemas

Praça General Tibúrcio 80

22290-270 Rio de Janeiro - Brazil

e-mail: anamoura@ime.eb.br

## Abstract

The development of emerging database applications requires a unified environment that should include, among others: a model which provides abstract object types constructors (structural and behavioral) and extensible types to support multimedia applications; powerful tools to help the user in the difficult task of modeling his/her applications, taking into account its schema evolution; and also a tight integration between the application language and database constructors to allow run-time development for database applications. In this paper we present an approach to keep the database integrity during the schema evolution process in an Object Management System, a research prototype of next - generation database system intended to meet the requirements of emerging database applications.

**1- Introduction**

The integration between the concepts of object oriented (O-O) programming language and the database management system has proved to be well-suited to such data-intensive application domains as Office Automation with hypermedia documents, AI, Geographic Information Systems and CAD/CAM. This technology is a fact, resulting of a countless number of commercial systems in the market, with report success: ONTOS, O2, Versant, ObjectStore, and others [KHOS93].

SIGO is an Object Management System (OMS) prototype in development at the Computer Science Department at IME. It was conceived to support emerging database applications in a small configuration environment ( PC like), providing a set of tools to help the end user in the complex task of developing his/her application. This task includes, among others, the application conceptual modeling and the structural and behavioral consistency of objects, which must be automatically assured during the schema evolution process.

A first academic prototype has come out in march 93, and since then a new development phase has begun. From the prototype experiences, many of its features were improved, while others were completely redesigned. Among these we can mention:

- the O-O conceptual modeling interface that controls the database schema evolution process in order to maintain structural and behavioral consistency of the objects in the database when type definition changes in the schema. It is an important and difficult area of investigation, and as far as we are concerned, most of commercial OODBMS do not take into account the behavioral consistency [KHOS93] [SOLO92] [KIM95] [DITT95]. This means that the conceptual schema is not allowed to change after the database population. Being closely dependent of the system data model, it was necessary to define appropriate directives to cope with this process. They were expressed through a set of rules that are directly applied to the database during its conceptual modeling [GUIM95]. The conceptual schema includes also a set of active rules that are activated whenever the schema is updated (after the database population), in order to keep the object integrity;
- the object manager, that has acquired new physical structures and access methods. These storage structures are based on the extended relational approach [MOUR95].

In this section we briefly recall the fundamental concepts which are relevant for our discussion. The system configuration is presented in Figure 1, and its functional modules are briefly described as follows :

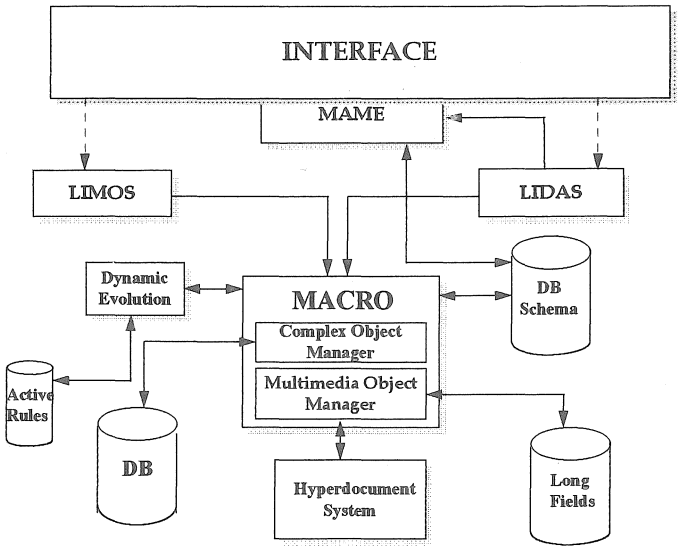


Figure 1: SIGO architecture

MAME is the user interface that provides the modeling of the real world objects. The modeling process is based on an O-O semantic network model, that will be described in 2. MACRO is the system module that takes care of the data placement, indexing and buffering. The storage structure is based on the extended relational model and the hierarchy structure is implemented according to the *class extension strategy* [WILL90] [MOUR95]. It means that object instances are stored in the class where they were defined. Each class is seen as an object, having a surrogate identifier, and it is mapped directly into a relation (or table). It maps the abstract object data model onto disk representation. MACRO includes also a special manager to deal with multimedia objects, providing an efficient storage and access retrieval [LIMF94]. The Schema Evolution module is based on active rules to provide schema evolution, a very important application requirement. LIMOS is the system interactive language [MOCO92] that provides queries onto the objects. It is based on SQL syntax and it does not provide encapsulation. LIDAS is the system programming environment to support user's applications development [MOUR95]. It generates automatically a set of methods in C++ for the object creation, retrieval, deletion and manipulation, the important step that provides the necessary requirements for the user to develop his/her applications. Finally, the hyperdocument system is a module that provides the ability to produce documents [QUAD95] that may be made up from some components of a conceptual schema or also from external multimedia components. Documents are presented taking into account temporal and synchronization aspects that can be specified through a special document editor, not presented in this paper.

The text is organized as follows: section 2 describes some of the main characteristics of the O-O data model, as the schema evolution integrity is closely tied to it; section 3 defines a set of meaningful directives to support schema evolution, exploring the use of active rules to keep the behavioral aspects of the database application. Finally, section 4 contains some concluding remarks about the system current state, and points out some issues for future work.

## 2- The Object Model Concepts

SIGO object model is based on a semantic network [ROMO 91], whose components may be represented through an hierarchical schema (Figure 2), where the *arc g* represents a generalization and the *arc a* an aggregation. Objects in SIGO are typed and they are associated to a class, being represented by a pair (identity, value). A class groups objects with the same properties (structure and operations), encapsulating data and behavior. Every user class defined in a conceptual schema is a subclass of the metaclass Classes that groups all the class structure in the system. Each class created in SIGO has a unique name and an identifier (*surrogate type*) generated by the system.

The model provides the possibility of including indexes over attributes and specialization constraints, defined obligatory in a subclass. It associates a condition to an inherited attribute (or attribute), meaning that during the subclass population the object will not be created unless the condition is satisfied by one of the inherited attributes. Similarly, primitive and constructor types are basic types and represent the main class components, as they are recursively used to form user AOTs (Abstract Object Types). Each primitive and constructor type (set, list, tuple) have their own group of operations. Primitive types (integer, string, real, boolean, data) offer the possibility to include two properties: "characteristics", to define if the attribute is unique (the same value is not allowed), not null, both or none (without restriction); and "constraints" that are directly applied to the attribute and concatenated by the "OR" operator, when necessary. Three sorts of constraints are provided: value, range and state, meaning that the user may respectively define to the attribute a specific value, a range or a set of values.

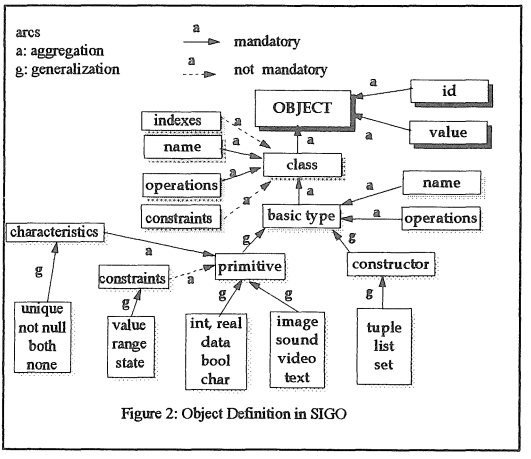

Figure 2: Object Definition in SIGO

Long types are available in the system and they also include a list of intrinsic functions that provide multimedia object capture and retrieval, concerning image, sound, video and text. Inheritance process in SIGO is total, i.e, a subclass inherits all the properties of its superclasses. It supports multiple inheritance, having specific

rules to solve conflict problems. The system does not provide any specific data definition language, as the conceptual schema definition is done through a friendly O-O interface named MAME, presented in Figure 1. It allows the user to define his/her classes by specifying their attributes, methods and constraints.

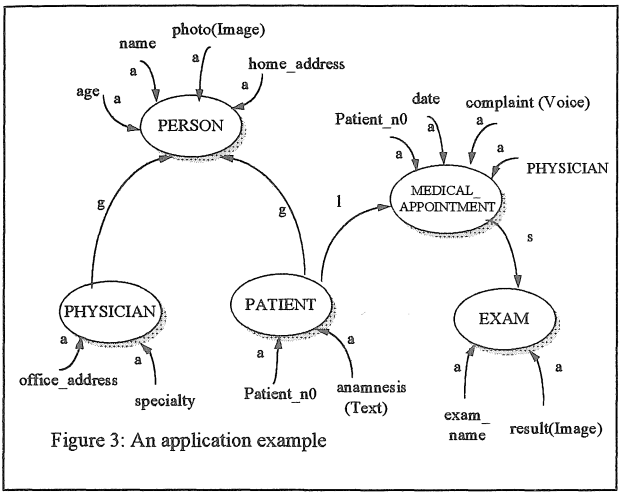

Figure 3: An application example

Figure 3 describes a simple application (including multimedia objects), represented in the SIGO object model. Person is an aggregation (*arc a*) of photo (image type), age(integer), name(string) and home_address(string). Physician and Patient are subclasses of Person and consequently inherit all the Person properties. Similarly, class Physician is represented by the attributes salary (real), specialty (string), and office_address (home_address type); and class Patient by the attributes patient_n0(integer), patient_story(Text) and a list (arc l) of occurrences of class Medical_Appointment, which is an aggregation of date(string), complaint(voice), Physician(Physician type) and a set of (arc s) occurrences of type Exam, which is also another class, constituted by the attributes exam_name(string) and result(image). All the classes defined in the conceptual schema may become automatically persistent with the user agreement.

SIGO conceptual model provides a rich interface that allows the user to specify explicitly and interactively many requirements (constraints over classes and attributes, indexes and key properties) not present in most existing OODBMS as already mentioned by [DITT95]. In $O_2$ [BANC92], Objectstore [SOLO92], ORION [KIM90], for example, these specifications may be specified by the use of methods.

### 3- The Schema Evolution Process

One of the important requirements of the O-O applications is schema evolution, i.e, the ability to dynamically make a wide variety of changes to the database schema[BANE87]. Most of the OODBMS commercially available today take into account only the structural consistency of objects, not allowing changes in the schema after the database population.

### 3.1- Rules for Schema Evolution

The schema evolution process includes two important aspects regarding the consistency of intentional and extensional database components. These aspects are controlled by mechanisms that ensure the database integrity as a whole. Zicari in [ZICA91] defines consistency as follows:

- Structural consistency: refers to the static part of the database. It assures the correction of the database schema graph, i.e, it must be represented by a direct acyclic graph, as well as it ensures that the component description of its components (attribute and method name definition, attribute and method scope rules, attribute types and method signatures) are compatible.
- Behavioral consistency: refers to the dynamic part of the database. An O-O database is consistent if each method respects its signature and if its code does not result in run-time errors or unexpected results.

However, we consider in this paper another consistency category, named *instantiation integrity*. It assures that all the class instances satisfy their class definition. This aspect is maintained following one of the mechanisms [LECL88] [CATT91]: (1) *write-once* types (the system disallows type modification once instances have been created. This is the option taken by most OODBMS [KHOS93] [SOLO92]); (2) *lazy* or *deferring propagation* (the object changes are propagated to the database until they are requested) as in ORION [KIM92]; (3) *immediate update* (the object changes are propagated immediately to the database as in the case of the GEMSTONE [PENN87]; and through (4) versions (it maintains an historical view of the database during its evolution), as in IRIS [FISH87]. In this work, the evolution schema process is carried out over a unique schema using the alternatives (2) or (3), according to the update operation executed.

To assure the structural consistency schema some invariants were defined, following the taxonomy presented in [BANE87]. They were determined according to the system data model, where the word *property* in the context of this paper means the components associated to a class such as: *attributes, methods, characteristics, constraints* and *indexes* (see Figure 2).

The behavioral consistency, concerning the object and method integrity, is carried out by the use of active rules, that are associated to the schema update procedures. Active rules are database structures that are able to recognize specific events from some determined condition occurrences. In this case, these rules are triggered and consequently they execute some actions, according to the paradigm EVENT / CONDITION / ACTION, where the EVENT indicates when a rule is triggered; once it is triggered, the CONDITION is verified and the ACTION is executed whenever the condition result is true [CERI94]. The behavioral schema evolution (extended to the objects and methods) of each class in SIGO is possible according to the development of some procedures:

- specification of the active rules associated to the object update operation as a result of the schema update (updates extended to all sort of properties in a class), as well as the relevant actions concerning the methods directly affected by the changes (an action could be, for example, an attribute replacement);
- integration of these active rules to all update procedures that are possible in a conceptual schema in SIGO;
- the rules are created and stored in a special repository, named *active rules repository;*
- the strategies (2) and (3) mentioned before are executed depending on the update operation: the schema changes can be *immediately propagated* (when the operations directly affect the object structure in disk) or *lazily propagated* (when the object update does not modify the object structure in disk, the schema changes may be propagated to the objects only when they will be referenced by an update method or object retrieval, for

example). In fact, the activation of one of these procedures (update or retrieval operation) is the EVENT that triggers the action rules. Then the active rule conditions are evaluated in order to verify if the referenced object or method belongs to the updated classes. Finally, if the condition is evaluated as *true,* the object or the method is effectively modified, according to the ACTION defined by the rule in execution.

The procedures taken for the structural and behavioral consistency will be enumerated, taking into account some important directives [GUIM95]:

1. The graph corresponding to the schema database must be acyclic, directed and connected. All the classes are subclasses of the metaclass Classes.
2. All the classes have distinct names, and all the properties (inherited or defined in the class) are necessary distinct.
3. All the class properties have a known origin.
4. A subclass always inherits all its superclass properties.

Besides, some rules were provided to ensure the schema integrity after changes in a class:

- **Rules for Conflict Resolution**

**Rule 1:**

When a class creates a property whose name already exists in its superclass, the system will concatenate this new property to the name of the actual class.

**Rule 2:**

When different superclasses have properties with the same name and origin, they will be inherited only once.

**Rule 3:**

When a subclass has superclasses whose properties have the same name but distinct origins, the system will rename them, concatenating their names to their origin classes, so that all of them be inherited by the subclass.

- **Rules for Property Changes**

**Rule 4:**

A property cannot be modified outside its origin class, except in case of conflict resolution.

**Rule 5:**

When an attribute name or domain is changed or when it is deleted from its origin class, this change must be propagated to the methods and indexes that referenciate it.

- **Rules for Schema Manipulation**

**Rule 6:**

A class A can only be added to the superclass list of B if A is not subclass of B .

**Rule 7:**

In case the class A is removed from the superclass list of B and A it is the only superclass of B, then B will be made subclass of all superclasses of A .

**Rule 8:**

When a class is added to the schema its name must be unique.

**Rule 9:**

A class cannot be removed from the schema unless it is disconnected from the schema graph. A class is considered disconnected from the graph when it does not have neither at least two superclasses, nor locally defined attributes and when it is not component object of another class.

**3.2- Schema Manipulation**

This section describes the procedures that will be applied to a conceptual schema designed and populated in the SIGO environment, taking into account its structural and behavioral consistency.

**1. Changes in a class contents**

**1.1 Attributes:**

**1.1.1 Add an attribute to a class:** apply rule 1

In case of name conflict, the new variable will be renamed, i.e, its name will be concatenated to its class. Each object belonging to the class and subclass extension will be added with a value specified to this new attribute. This value may be constant or null if it is not pre-specified. This operation may generate temporary inconsistencies to the methods defined by the user's applications which reference methods not yet automatically generated by LIDAS [MOUR95].

**1.1.2 Remove an attribute from a class:** apply rules 4,5 and 9

This change can only be processed in the class where the attribute was defined (origin class) and it will be propagated to all its subclasses. Each object belonging to the class and subclass extension will lose the value associated to the removed attribute. Methods, indexes and constraints defined in the subclasses will also be affected:

- methods and indexes: those which referenced the removed attribute become invalid and the user is informed of this situation.

- constraints: in the case when an attribute is a component of a constraint, the subclasses of the modified class should eliminate that constraint.

**1.1.3 Rename an attribute:** apply rules 4 and 5

The class which has that attribute as component from now on will reference it by the new name. This change will be propagated to all the subclasses of the actual class. This operation has no effect over the objects. The effect of this operation over the methods is similar to 1.1.2.

**1.1.4 Change the attribute domain:** apply rules 4 and 5

The objects belonging to the class and subclass extension will be added with the values correspondent to the new domain, according to the Table 1. Nothing happens to those objects which reference another object by an identifier (oid). When the object corresponds to a value, a conversion must be explicitly done. Similarly to the case 1.1.1, a null value may be given to the object if another one has not been specified.

Methods, indexes and constraints will be affected by this change in the following way:

-methods: those which reference the modified attribute become invalid and the user is informed of this situation.

-indexes: the attribute is removed from the indexes where it appears as component. If it is the only component of an index, this one is also removed from the class.

**1.1.5 Change constraints and characteristics of an attribute:** apply rule 4

This change will not affect methods and indexes that have this attribute as component. Not all the constraints and characteristics update can be executed in the system. The possible changes are showed in the Table 2 and 3, respectively:

**Constraints**: the objects belonging to the class and subclass extension may have a new value to the attribute if they do not satisfy the new constraint. This value may be specified or a null value may be associated to the attribute.

**Characteristics:** as seen in Figure 2, a characteristic is directly associated to an attribute, that may be unique, null, etc. If one of these attribute properties change, all the objects belonging to the class and subclass extension may be modified according to the new characteristic: if the objects satisfy the new characteristic, nothing happens; otherwise a new value is specified for them.

**1.2 Methods**

**1.2.1 Add a method to a class**:

The same as for attributes (1.1.1). This operation has no effect over the other methods defined in the class.

**1.2.2 Remove a method from a class:** apply rule 4

This operation may result inconsistencies to those methods which reference the one removed.

**1.2.3 Rename a method:**

The same as for attributes (1.1.3). The effect over the methods is similar to 1.2.2.

**1.3 Indexes**

**1.3.1 Add an index to a class**

The same as for attribute(1.1.1), now considering the index name. An index structure will be created over the attributes belonging to the objects in the class extension. This operation has no effect over the methods defined in a class.

**1.3.2 Remove an index from a class**

This change is only possible in the origin class where the index was defined. There is no effect over the objects in the class and subclass extension, as well as over the methods.

**1.3.3 Change the name of an index**

The same as for attributes (1.1.3). Due to the access and storage structures used in the system (Paradox Engine v. 3.1, Borland), this operation is done in two steps: the existing index structure is removed and a new one is created with the new name according to 1.3.1. This operation has no effect over the methods defined in a class.

**1.3.4 Add an attribute to an index**

The attribute must be defined in the index of the origin class. The effect of this operation over the objects is similar to 1.3.3. This operation has no effect over the methods defined in a class.

**1.3.5 Remove an attribute from an index**

The same as in the previous situation. The effect of this operation over the objects is similar to 1.3.3. This operation has no effect over the methods defined in a class.

**2. Changes in the class hierarchy (generalization arc)**

**2.1 Make a class S into a superclass of a class S1:** apply rules 2, 3 and 6

This change cannot happen if the class S is a subclass of S1. Otherwise the set of S1 properties will be increased with those of class S. The values for the inherited attributes may be defined as in 1.1.1. This operation has no effect over the methods defined in the class S1.

**2.2 Remove a class S from the superclass list of the class S1:** apply rule 7

If S is the only superclass of S1, S1 superclass list will be added to the superclasses of S. The properties of S1 which were directly inherited from S will be removed. If the subclasses of S have any class constraints including the removed components then they will be also removed. This change will be propagated to all subclasses. For those attributes which are inherited from the class S the effect of this operation over the objects is similar to 1.1.2. Figure 4 illustrates this situation. This operation has no effect over the methods defined in the class S1.

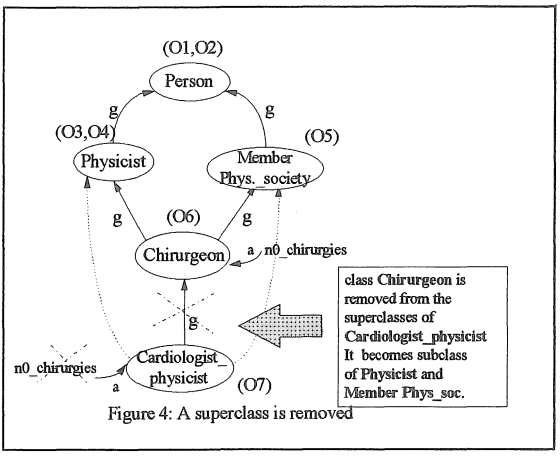

Figure 4: A superclass is removed

**3. Changes in a class**

**3.1 Create a new class:** apply rule 8

A class is not effectively created in the system unless it has a defined attribute or it is a subclass of two or more classes. This operation has no effect over the objects in the class and subclass extension, as well as over the methods.

**3.2 Remove a class from the schema:** apply rules 7 and 9

A class that is object component in another one cannot be removed. If this situation does not happen, the procedure 2.2 is applied. Being completely disconnected from the graph the class can be dropped from the schema. This operation results on the removal of all the existing objects in the class being removed. The effect of this operation over the methods is similar to operation 2.2, extended to all the subclasses of the removed class.

**3.3 Rename the class:** apply rule 8

This operation has no effect over the objects in the class and subclass extension, as well as over the methods.

**4. Migrating an object from a class O(origin) to a class G (goal)**

The effect of this operation over the objects is similar to the removal and inclusion of an object, although two constraints must be taken into account: if an object is referenced by another one it cannot migrate; in the migration process the objects keep their identifiers (oids). The migrated objects may keep their values, they may receive an initial constant value or a null one (this can happen also for the inherited attributes) or they can even lose some attributes. This procedure is illustrated in Figure 5. This operation has no effect over the methods defined in the class.

**4- Conclusions**

This paper presented some aspects of an object manager system in development at the Computer Science Department at IME, placing emphasis to the schema evolution process. Some directives were established to maintain the schema integrity throughout the schema evolution process, even after the database population. These directives have taken into account the structural and behavioral aspects, according to the system O-O data model. The behavioral consistency is based on active rules, that are activated whenever an update operation is executed. The SIGO prototype is being implemented in C++ on Windows environment for PC. At present we are investigating the use of active rules to support the definition of general rules (declarative and conditional ones) during the conceptual schema definition.

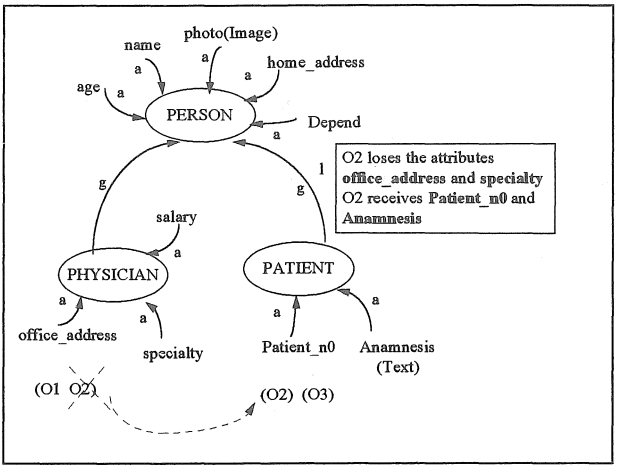


Figure 5: An object Migration

**Acknowledgments**

The authors wish to acknowledge the financial support from the Brazilian CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico and FAPERJ - Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro.

**References**

[BANC92] F. Bancilhon, C. Delobel, P. Kanellakis, "Building an Object - Oriented Database System - The Story of O2 ". Ed Morgan Kauffmann Publ. 1992.

[BANE87] Banerjee J., Kim W., "Semantics and Implementation of Schema Evolution in Object-Oriented Databases", ACM SIGMOD Record, vol . 16, N° 3, 1987.

[CERI94] Ceri,S. et al. Active Database Systems. Database Workshop on New Database Research Challenges, PUC-, Rio de Janeiro, sept 1994.

[CATT91] Cattell, R.G.G. "Object Data Management - Object Oriented an Extended Relational Database Systems". Addison-Wesey Pub. Comp., 1991.

[DITT95] Dittrich A.K, Dittrich K.R. "Where Object-Oriented DBMSs Should Do Better: a Critique Based on Early Experiences". In Modern Database Systems- The Object Model, Interoperability and Beyond, by Won Kim (ed). ACM Press, chap. 12, p. 238- 254, 1995.

[FISH87] Fishman, D.H. et al. "IRIS: an Object-Oriented Database Management System". ACM Transactions on Office Information Systems. Vol. 5, $n^0$ 1, Jan 1987.

[GUIM95] Guimarães, R. N., "Interface Orientada a Objetos para Criação e Manipulação de Esquemas Conceituais no SIGO", Master Thesis, Instituto Militar de Engenharia, March 1995.

[KIM90] Kim, W. et al. Architecture of the ORION Next-Generation Database System, IEEE Trans On Knowledge and Data Engineering, vol 2 n0.1, March, 1990.

[KIM92] Kim, Won. "Introduction to Object-Oriented Databases". MIT Press, 1992.

[KIM95] Kim W. "Object-Oriented Database Systems: Promises, Reality and Future". In Modern Database Systems- The Object Model, Interoperability and Beyond, by Won Kim (ed). ACM Press, chap. 13, 1995.

[KHOS93] S. Khoshafian, "Object - Oriented Databases", ed. John Wiley & Sons Inc. 1993.

[LECL88] Lecluse, C. et al. "Object-Oriented Database Systems". ACM SIGMOD Proceedings, March 1988.

[LIMF94] Lima C., Moura A.M.C., Ferreira P.C.C. "Multimedia Object Management in an O-O Database Environment". Computer Research 2, edited by R. Baeza, Plenum Press, 1994.

[MOCO92] Moura A.M.C.,Costa M.R., "An Interactive Query Language for an Object Oriented Database", XII International Conference of the SCCC, Santiago, Chile, Oct 1992.

[MOUR95] Moura, A.M.C. et al. "Dealing with Persistence in an Object Manager System Using Relational Implementation. XV International Conference of the Chilean Computer Science Society. Arica, Chile, Nov 1995.

[PENN87] Penney D. J. , Stein J. "Class Modification in the Gemstone Object-Oriented DBMS, SIGPLAN Notices, Dec 1987.

[QUAD95] Quadros, J.T. "Modelagem de Hiperdocumentos em um Sistema Gerenciador de Objetos". Master thesis, IME - Rio de Janeiro, in development.

[ROMO91] Rossetti L., Moura A.M., "Modelagem de Esquemas em um Ambiente De banco de Dados Orientado a Objetos", XVII Informatics Latinamerican Conference (CLEI 91), Venezuela, Caracas, Jul 1991.

[SOLO92] Soloviev V. "An Overview of Three Commercial Object-Oriented Database Management Systems: ONTOS, Objectstore and $O_2$". SIGMOD Record, Vol 21, $n^0$ 1, Mar 1992.

[WILL90] Willshire M. J, Kim H.J. "Properties of Physical Storage Models for Object-Oriented Databases". IEEE Transactions on Software Engineering, 1990.

[ZICA91] Zicari,R. "A Framework for Schema Updates in an Object-Oriented Database System", Proc. of the 7 th IEEE International Conference on Data Engineering, April 1991, Kobe, Japan.

*Table of Possible Convertions Among Domains*

| | PRIMITIVES | | | | | CONSTRUCTORS | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| DOMAIN | INT | REAL | BOOL | CHAR | DATE | LIST | TUPLE | SET | CLASS | OBS |
| NEW DOMAIN | | | | | | | | | | |
| INT | XX | yes | yes | yes | yes | no | no | no | no | |
| REAL | yes | XX | yes | yes | yes | no | no | no | no | (4) |
| BOOL | no | no | XX | yes | no | no | no | no | no | |
| CHAR | yes | yes | yes | XX | yes | no | no | no | no | (4) |
| DATE | yes | yes | no | yes | XX | no | no | no | no | (4) |
| LIST | yes | yes | no | yes | yes | XX | no | yes | yes | (1) |
| TUPLE | yes | yes | no | yes | yes | no | XX | no | yes | |
| SET | yes | yes | no | yes | yes | yes | no | XX | yes | (2) |
| CLASS | yes | yes | no | yes | yes | no | yes | no | XX | |

Table 1

*Table of Possible Convertions Among Constraints*

| CONSTRAINT | NONE | VALUE | RANGE | STATE | OBS |
|---|---|---|---|---|---|
| NEW CONSTRAINT | | | | | |
| NONE | XX | yes | yes | yes | (3) |
| VALUE | yes | yes | yes | yes | (4) |
| RANGE | yes | yes | yes | yes | (4) |
| STATE | yes | yes | yes | yes | (4) |

Table 2

*Table of Possible Convertions Among Characteristics*

| CHARACTERISTIC | NOT NULL | UNIQUE | BOTH | NONE | OBS |
|---|---|---|---|---|---|
| NEW CHARACTERISTIC | | | | | |
| NOT NULL | XX | yes | yes | yes | (4) |
| UNIQUE | no | XX | yes | no | |
| BOTH | no | yes | XX | no | |
| NONE | yes | yes | yes | XX | |

Table 3

(1) In the domain modification from set to list the elements will be sorted or the list will be initialized with a value specified by the user;
(2) In the domain modification from list to set the duplicated elements will be deleted;
(3) Keeps the values of the objects;
(4) In case the existing values do not correspond to the new domain, constraint characteristic, the user should inform the new values.