A Command and Control System Based on a Multi-Media Object-Oriented Data Base and a Logic Programming Language

Jean-Paul A Barthès
Université de Technologie de Compiègne
Computer Science Dept.
C.N.R.S. UA 817
BP 649
60206 Compiègne Cedex - FRANCE
Tel: 33-44-23-36-37

Yannick Le Noan
Graphael
255 Bear Hill Road
Waltham, MA 02154
617-890-7055
BP 649 617-890-7055
C.N.R.S. UA 817 Waltham,MA 02154
Computer Science
Université de Technologie de Compiègne

Abstract

Multi-media and object-oriented data bases are used as information storage and retrieval systems in a large number of industrial and military applications. Thanks to a marriage of Data Base management technology with AI concepts, the French Navy is today exploiting the potential of both to produce a system that monitors the worldwide movement of ships of many nations.

The system is based on a multi-media object-oriented data base management system, G-BASE™, and a logic programming language, G-LOGIS™. This paper discusses VORAS, the multi-user prototype of the data base developed in a research context and G-BASE the commercially available product for workstations. We report how we used the simple and powerful model of recursive frames to implement an integrated system in VORAS and how the French Navy Command and Control application was developed using G-BASE and G-LOGIS.

G-BASE™ and G-LOGIS™ are trademarks of Graphael

Introduction

As artificial intelligence techniques are applied to an increasing number of realistic situations, new problems appear over the proper handling of large quantities of multi-media information. Data/knowledge bases are needed to provide permanent storage for persistent objects, as well as efficient access for multiple simultaneous applications--expert systems usually work on shared data [14]. Object-oriented data bases (see in particular "Expert Data Bases," Kershberg [21]) appear to be well suited to solve such problems. We describe hereafter two concurrently developed implementations of the same model of an object-oriented data base: the first one, VORAS, is a multi-user prototype developed in a research context, and the second one, G-BASE, is commercially available for workstations.

Several other attempts have been made to solve similar problems [15], [23], [27], [28], [31], [33], [35] which has lead to products, prototypes, or simply interesting mechanisms (see also "Modeling Complex Objects for Multimedia Data Bases," Adiba [3]). However our approach is somewhat different from previous attempts as will be shown. The products main features are:

- use of a simple, yet powerful, model of recursive frames to structure the objects, inheritance, implementing abstract types, and procedural attachments,
- use of permanent storage, with multi-user concurrent access,
- use of separate LISP environments at the user level for storing objects and methods,
- use of PROLOG logic extended with unification of objects for querying the Knowledge Base,
- use of indexing to deal efficiently with millions of objects,
- use of special secondary storage format for improving compactness and access time.

This paper reports the results of our efforts to integrate data base mechanisms and AI concepts. The first part briefly presents the representation formalism we used and discusses how we solved the traditional data base requirements such as permanence and sharing in the AI environment. The second part describes how using G-BASE, the French Navy can now permanently save a large number of multi-media objects (the application today has several hundred thousand objects).

1. The VORAS and G-BASE Representation Formalism

1.1 Overall approach

The difficulty of the problem is the fact that two conflicting approaches needed to be combined: AI, where a dominant factor is the prototyping paradigm; and the data base world, where models represent strict specifications.

We take the global approach of building an entire object data manager capable of handling a large number of shared objects to avoid the so-called problem of "impedance matching" between the AI environment and standard data bases. Contrary to Rowe [27] but like Dittrich et al [11], we believe that traditional DBMS are not suited for handling objects efficiently, and that the traditional data base functionalities must be reconsidered and reimplemented in the context of object handling. Thus we differ from approaches extending conventional data base models like POSTGRES, or even those using background Smalltalk environments such as OPAL/Gemstone [15], [23]. We also differ from approaches translating standard representations into frames like Chow [9], or using the E-R model for defining relational schemas such as TSR [20].

1.2 Representation: the Property Driven Model

The PDM1 (Property Driven Model) was developed at the University of Compiègne in the late seventies [6], a period of time strongly influenced by Minsky [24], and also by Abrial [2]. PDM is a representation using Recursive Frames.

Objects in PDM are represented simply as attribute-values pairs. Some attributes have immediate associated values and are called terminal properties; others are links between objects and are called structural properties. Attributes, or properties, are in

---

1 Not to be confused with the more recent PROBE Data Model (Manola 86)
PDM has some specific features:
- Contrary to most models, values associated with terminal properties are not restricted to any specific type, and can thus be anything from a string to a picture or an executable function (which is useful for implementing demons). However they can be restricted by the use of demons attached to the specific terminal property thus implementing data types. Such demons are invoked at creation or modification time (formatting demon) or for displaying values.
- **Terminal properties may be inverted**, creating external references to objects called entry-points. Such inversions occur at creation time by an appropriate demon. Entry-points are automatically maintained and updated when values are modified. **Entry-points are not required to be unique**, which simplifies indexing.
- **Comparison operators** may be user-defined and associated with terminal properties.
- **Structural properties** are automatically inverted, which allows one to navigate easily within the data base, and avoids extra indexing structures as in [23].
- **Entities** (all objects, classes and instances) have an internal unique identifier, their internal key. Such an identifier is otherwise called "oid" [27], OOP [23], or UID [28].
- Any object can grow or shrink at any time.
- **Classes** are modeled as objects providing an integrated data dictionary. Objects are instantiated using the classes in a similar way as Smalltalk.

PDM is a consistent object system, i.e. every object has a unique identifier, and there are no dangling references.

**Inheritance Hierarchies**: As in most frame systems classes belong to an inheritance hierarchy that is orthogonal to the three levels of objects: instances, classes, and metaclass. This is shown on Fig 1.

![The PDM three levels](image)

**Indexing**: When dealing with several hundred thousand instances, objects need to be indexed so that they can be found efficiently without knowledge of their internal names. This is done by using entry points (called keys in the standard data base terminology.) Any field corresponding to a terminal property and containing a value can be inverted in as many ways as desired by specifying an appropriate demon in the corresponding property slot. When the field is given a value, an entry point is created (or updated) dynamically as an index. Contrary to traditional data bases, there is no need to define such things as primary or secondary keys. All fields are created equal and can be inverted in many different ways. Keys are not even required to be distinct, i.e. the same key can point at different objects.

**Storing Objects**: Traditional Frame Representation Languages found in the artificial intelligence field do not use secondary storage, but instead save core images in between sessions. Since we were interested in dealing with large volumes of data we decided that loading and saving the whole content of the data base at the beginning and at the end of a session was not acceptable, especially in a multi-user environment. Although mechanisms such as MOBY remove the size limit on virtual memory, they do not ensure data consistency at all times. Like Thatte [30] we think it would be a pity not to use the rich structured environment of LISP. However, we do not think that designing a persistent virtual memory will solve all problems. For one thing, flat versions of objects are needed for exchanges between heterogeneous machines or for exchanges with more traditional data base systems. Also when the system becomes really large, data is scattered throughout virtual memory (there is no single clustering criterion which can guarantee locality for all queries since objects are composite), and one can expect the number of page faults to be of the order of the necessary number of disc access.

**Dynamicity**: A strong feature of the VORAS system is its dynamic aspect. At any time the user is free to modify objects by adding or removing properties. Adding properties does not create any problem, although there is no mechanism to allow the user to update already existing objects. Old objects are displayed as they were and can be updated if progressively needed. When a property is removed from a model, one must take care of modifying the procedural attachments involving such a property accordingly. Otherwise no changes are made to the data and the values in the instances simply become unavailable or "forgotten”. However, to prevent the user from potentially dangerous modifications of the data base model, we restricted access rights to the object models, leaving the responsibility to the specialist (e.g. data base administrator).

**Implementation**: The VORAS and G-BASE systems
A frame editor was first developed as a set of LISP functions to handle data according to the PDM model. The name VORAS was coined from the various operations one could perform upon the data: View (query), Order, Replace, Add, Suppress. The data base management system, VORAS, was designed to free the user from programming the common maintenance and retrieval functionalities. All programming, if any, is done by the Data Base Administrator who writes and debugs demons (procedural attachments). As a consequence the system is highly interactive with a somewhat passive user. G-BASE is the industrial version of VORAS. G-BASE includes a more sophisticated user interface, as well as a programming environment. An interesting feature of the approach is that the same functions are used for creating instances of objects at the end-user level, as for building models at the specialist level (e.g. data base administrator), which makes it more compact. Also, since models are themselves objects and can be handled as such, this means that we have an "integrated dictionary", since our specialist, the data base manager, is allowed to manipulate models as ordinary objects.

**Languages found in the artificial intelligence field do not use secondary storage, but instead save core images in between sessions.** Since we were interested in dealing with large volumes of data we decided that loading and saving the whole content of the data base at the beginning and at the end of a session was not acceptable, especially in a multi-user environment. Although mechanisms such as MOBY remove the size limit on virtual memory, they do not ensure data consistency at all times. Like Thatte [30] we think it would be a pity not to use the rich structured environment of LISP. However, we do not think that designing a persistent virtual memory will solve all problems. For one thing, flat versions of objects are needed for exchanges between heterogeneous machines or for exchanges with more traditional data base systems. Also when the system becomes really large, data is scattered throughout virtual memory (there is no single clustering criterion which can guarantee locality for all queries since objects are composite), and one can expect the number of page faults to be of the order of the necessary number of disc access.
reads. In addition the size of the virtual memory space may not be large enough on some current machines to accommodate the total number of objects. Finally, as noted by Stonebraker [29], the price of restructuring becomes small with the increase of processing power, compared to the price of disk accesses.

Consequently we decided to isolate data from the runtime environment, and to store the objects on dedicated physical space in a special flat format, designed to minimize the restructuring time of LISP objects. A detailed discussion of the corresponding formats is outside the scope of this paper and can be found in [4]. All objects (and functions) are stored in the same linear space. Internal object identifiers (keys) are used for linear hashing. Secondary memory is packed periodically (every month for some of our applications). It is possible to use more elaborate memory management techniques such as in EXODUS or in POSTGRES [29], but they lead to quite complex object structures.

**Sharing Objects** The most difficult problems when using frames occur when data must be shared by many users.

One has to distinguish between elementary (single non-composite object) operations, and sequences of operations. During elementary operations the internal state of the data base may be physically inconsistent (during modifying commands), and in such a case all accesses must be prevented from other users. For read only accesses, concurrent reads are permitted.

To preserve logical data integrity, concurrent updates are serialized by grouping consistent updates as transactions, expressed as a series of elementary operations. However, since we are dealing with networks of frames and of composite objects, some operations must be deferred until the actual execution of the transaction. Furthermore, as noted by Skara *et al* [28], traditional concurrency techniques are not appropriate for objects systems, specifically when long interactive sessions are involved. Consequently VORAS uses a deferred update approach. It consists of synthesizing an update program using temporary variables, taking advantage of the LISP environment. The update program which may include user-defined functions is then executed during a transaction within a consistent environment while the disk is locked. In doing so we avoid long waits or aborts mentioned in [28] that could result from the high interaction between objects during long interactive sessions.

**Note:** G-BASE implements a two-phase locking mechanism for sharing objects.

**Object-Oriented Programming** G-BASE objects incorporate object-oriented programming techniques, including the support of multiple inheritance for classes and properties, the ability to have associated methods represented as objects, and the use of the "send" function as part of the programming interface. Details are given in "Introducing Object Paradigm in OODBMS," Lenain and Barthes [22].

**Browsing** Since structural properties are fully inverted in PDM, they provide easy navigational access. Furthermore, browsing in a semantic data base provides an efficient user-interface as mentioned by Rogers and Cattell [26]. A screen copy (Fig 2) shows data in graphical form, another one (Fig 3) shows picture data and drawings being displayed. Pictures are not normally stored on external media as in ORION [35] but can be if needed.

---

**Figure 2** The Graphic Representation of Objects

128
Queries In VORAS and G-BASE the query mechanism has several layers:

- At the lowest level, objects are indexed using entry points discussed previously. Hence, if one asks the question
  Washington?
  the system is able to answer directly
  "I have 2 persons, a city and a ship named Washington"

Furthermore the time necessary to answer the question does not depend on the number of objects in the data base.

- At a higher level VORAS supports a query mechanism. Formal queries are used internally or may be synthesized using an interactive interface. The query processor is optimized to perform the minimum number of disc reads.

For current applications, queries are canned and offered as a series of menus.

G-BASE uses case grammars in predictive mode to present the user with legal choices in active menus, allowing him to construct queries interactively.

- At still a higher level, a free language interface was developed that reconstructs formal queries from open text.

Deductive Accesses Recently an interface, G-LOGIBASE (renamed PRO-QUERY) [17] was built between G-BASE and the object-oriented PROLOG system, G-LOGIS [18]. It can be used to define queries, as well as relations as in PROLOG predicates. It is thus possible to define virtual relations, then use them inside queries. G-BASE attributes are transformed into predicates that can be used in the unification process. The system was used to build an integrated expert system in the French Navy application.

Figure 3 Prototype of the Command and Control System for the French Navy

2. The French Navy Command and Control System

In support of the French Navy's mission to protect the sea lanes of the Mediterranean, Graphael has developed an intelligent data base that monitors the movements of many ships and provides the access of this information to Naval commanders on land and sea. The system provides the kernel for an expert system that tracks a large number of vessels, detects significant evolutions in the operational theatre, and constantly reevaluates the tactical situation in this dynamic environment.

2.1 The Multi-Media Data Base

The first part of this project was to present, in an intelligent way, all the information available. The SYCOM (SYsteme de Commandement de la Marine) is comprised of a data base that contains characteristics and situation information regarding thousands of maritime vessels, including activity, mission, availability, and position. It enables the the situation at sea to be analyzed by Navy personnel on land or at naval headquarters. G-BASE was instrumental in organizing and managing a subset of the system's 200,000-object data base. See Figure 3.

Application code contains both historical and current information as obtained through sightings, messages, and satellites. With the underlying data--including vector images and photographs--stored as objects, the many relationships can be captured and updated regardless of type. New relationships can be created and added at any time. G-BASE's multiple inheritance capability means new classes of objects can be created based on features of existing classes without completely rewriting code. As they are implemented, expert system functions are written in G-LOGIS.

Figure 3 Prototype of the Command and Control System for the French Navy
The operational situation is presented as a vector graphic of the Mediterranean region with graphic representations of all the naval vessels of interest or consequence. The mouse-sensitive chart allows officers to perform a wide variety of functions, such as pointing to any vessel graphic to obtain course and speed intercepts or straight-line trajectories to any vessel on the screen. This hypertext capability also allows access to any further information on the vessels, including a complete operational history.

Operators may also zoom in for a closer look at the situation. Not only can officers receive information about vessels, but also detailed information about prevailing currents and winds in the area, and expanded graphics of straights, hazards to marine traffic, and ports and docking facilities.

In addition, the system provides three query interfaces, including "Ships", "Messages", and "Radars", and powerful querying tools. As the user builds up a menu of queries in the course of seeking specific information, he may edit any one of the earlier queries without losing the line of inquiry. The information received through these queries can be presented as part of the geographic representation, or through windows on the screen. These windows contain data, text, vector images, and photographs.

The ability of G-BASE to store multi-media information made the development of this powerful interface possible.

2.2 The Expert System

The second part of the project was to develop the expert system, SEISM (Système Expert en Interpretation de Situation Maritime). SEISM is designed to detect and interpret significant evolutions of the operation theater using updated positions of the vessels. The information in G-BASE is accessed and the rules are formalized through G-LOGIS, Graphael’s logic programming language.

The Object-Oriented Logic Programming Language: G-LOGIS

G-LOGIS, developed by Paul Y Gloess at the University of Compiegne, is an extension of prolog logic to objects within the Lisp environment. It enables the user to mix each type of programming:
- Logic programming (Prolog)
- Functional programming (Lisp)
- Object-oriented programming with instantiation, inheritance and methods.

The order I inference engine of G-LOGIS enables the user to define logical relationships between classes or objects. G-LOGIS works on structured objects. The expertise rules are formalized after the description of the field of expertise in term of objects. G-LOGIS is based completely on an object-oriented language that allows any element of the Lisp environment to receive messages. Each object has its own methods that allow it to be unified with other objects. This implementation allows easy extensions of the unification and inference mechanism. G-LOGIS standard backtracking strategy can be extended to add specific treatments or other strategies; it also provides a blackboard facility and allows dynamic insertion and deletion of rules and objects.

The Expert Data Base

The use of two representation concepts to describe knowledge adds several advantages to both classic DBMS and classic expert systems development tools. Ordinarily, when relationships between objects are represented in rules, the system must review a large number of rules to find a specific link between two objects. In G-BASE, the relationship between the ship "Clemenceau" and the nation "France" is explicitly described within the object ship. Furthermore, the expert system developer can use data base information as objects of the environment. The data base itself is on the disk and the objects are loaded into virtual memory for access.

To enable users to access G-BASE information from G-LOGIS, several functions have been defined:

- (is-G-BASE *mobile *M)
- (nationality *N)
- ($type (Aircraft-carrier))

Notes: "*M" and "*N" are the variables, "$mobile" is a G-BASE entity class, "$nationality" and "$type" are properties of this entity class, "Aircraft-carrier" is a constant.

So in this predicate, "*M" is a mobile (stored in the data base) of nationality "*N" and of type "Aircraft-carrier".

- ($nationality *M *N)

Specifies a relationship between two objects.

If only one property is used in a premise of a rule, it is not mandatory to declare that "*M" is a Mobile and "*N" a nation. If "*M" and "*N" have values then a verification is made; if one is missing then it is calculated.

A few rules:

The syntax of a rule in G-LOGIS is:

$$\text{(-}) \quad (\text{premise-n})$$

1. Reaction to a specific event:

$$\text{(-}) \quad \text{reaction *ship *event}$$

$$\text{(known-event *event *place *observed-nation)}$$

$$\text{(going-to *ship *place)}$$

If there is a ship (*ship) of the observed nation (*observed-nation) and a known event (*known-event) of this nation somewhere (*place) and if this ship is going to this place then there is a reaction of the ship to this event.

2. Replacement of a ship

$$\text{(-}) \quad \text{replacement *arriving-ship *leaving-ship}$$

$$\text{($nationality *arriving-ship *nation)}$$

$$\text{($type *arriving-ship *type)}$$

$$\text{($nationality *leaving-ship *nation)}$$

$$\text{($type *leaving-ship *type)}$$

$$\text{(end-of-mission *leaving-ship)}$$

Note: end-of-mission is defined by another rule and depends on the time the ship stayed at one place as well as additional information.

If two ships (*arriving-ship and *leaving-ship) of the same nationality (*nation) and the same type (*type) and if one of the ships (*leaving-ship) is ready to leave (end-of-mission) the the arriving ship is a replacement.
Conclusion
We presented two systems, VORAS and G-BASE, for storing a large number of objects in a LISP environment, saving them permanently in secondary storage and providing shared accesses. Objects use a simple, flexible and powerful model of recursive frames called the Property Driven Model. A number of mechanisms have been implemented such as browsing, complex queries, object-oriented programming, and deduction leading to a desired result.

In utilizing G-BASE, the French Navy has discovered two unique advantages of the product. These advantages are as follows:
1. G-BASE manages all information available and presents it in an intelligent way;
2. The Expert System developer can use the Data Base objects and relationships without having to build and manage a data base.

Many other applications have been done using the two systems. VORAS was used to define a data base in Enzyme Technology, to implement a library catalog [5] (around 250,000 objects), in order to develop a complex planning system SEDLEX [31]. Rules and facts are written using the PDM format and saved permanently over long period of time. VORAS was also used to integrate parallel expert systems with a data base for simulating a heterarchical robot system [19]. G-BASE is being used in many military and industrial applications such as a large documentation management system at ALCOA [32].

REFERENCES


