Astronomical Data Analysis Software and Systems IX
ASP Conference Series, Vol. 216, 2000
N. Manset, C. Veillet, and D. Crabtree, eds.

SIMBAD as a Test Bed for two Object Oriented Database Management Systems: Objectivity/DB and O2

M. Wenger

Centre de Données astronomiques de Strasbourg, Strasbourg, France

F. Kinnar, R. Jocqueau

Haute Ecole de la Province de Liège Rennequin Sualem, Liège, Belgium

Abstract. SIMBAD is an astronomical database which gathers very heterogeneous information about astronomical objects, which makes it particularly suitable for Object Oriented Database Management Systems. And, as a matter of fact, the SIMBAD software was already build in-house around object oriented concepts more than 10 years ago.

It is therefore tempting now to develop a prototype of SIMBAD using available commercial ODMBS. Two prototypes were built independently with Objectivity/DB and O2 (ArdentSoftware), allowing us to explore these two database systems and compare their functionalities and performances.

The paper presents the results of this work. It shows through a few examples how the database design has to fit with the functionalities provided by each system. Queries using the current SIMBAD and the two prototypes are compared. More generally, the type of application suitable for each database system is discussed.

1. Introduction

The Object Database Management Systems (ODBMS) have been, for a few years, an emerging technology in astronomy.

On the contrary of Relational DBMS (RDBMS), which are well standardized (tabular data model, SQL language), ODBMS were born without any preexisting standard and can still be very different from one system to the other, in spite of the standardization efforts of the Object Data Management group (ODMG: `http://www.odmg.org/`).

The SIMBAD database (`http://simbad.u-strasbg.fr/`) has a simple data model, but contains heterogeneous data, both in type and quantity. The database software was developed at CDS, written in plain C, but already based on object oriented concepts, at a time when no commercial ODBMS was available. It has been operational since 1990. It is thus a good candidate for testing ODBMS, both because it is already based on Object Oriented concepts, and to prepare for a future system evolution. Two prototypes were built using **Objectivity/DB** (`http://www.objectivity.com`) and **O2** from Ardent Software

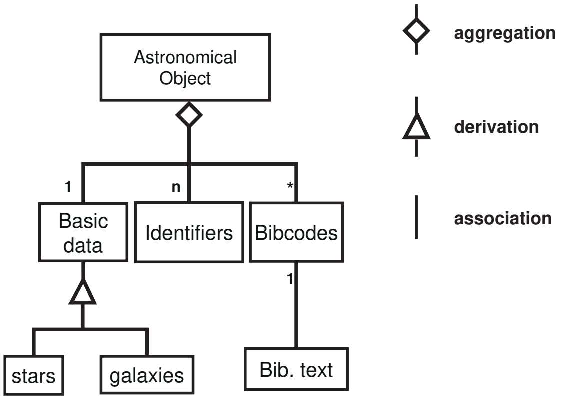

Figure 1. Data model schema for the prototype.

(`http://www.ardentsoftware.com`), these two ODBMS being already used in astronomy.

This paper will focus on a few examples showing that two ODBMS may behave very differently, and how this can lead to different implementations, closely related to the characteristics and the features offered by each system.

2. The Prototype

The prototypes had to be built in about four months, thus it was needed to simplify the actual data model as shown in Figure 1.

The prototype implementation includes the following modules which allow to measure the performance of the data base system in our context: a database loader, which should reveal problems linked with the initial loading of a database; query by identifiers, allowing to test indexing; query by coordinates, leading to experiment clustering and sequential access; and query by bibcodes, giving the opportunity to check the association feature of the ODBMS.

3. Objectivity/DB and O2

In spite of an announced ODMG compliance, Objectivity/DB and O2 remain different. If they share a lot of commonalities, like the same fat client/page server architecture, they have many differences:

If O2 has a dynamic schema allowing to add and modify classes and methods at run time, Objectivity/DB is more static, allowing some run time interaction with the schema only since its last release.

O2 has its own development language, 02C, whereas Objectivity is only interfaced to the regular languages C++, JAVA and Smalltalk. O2 uses OQL as standard query language. Objectivity proposes SQL++ as an option. O2 has also an extended development environment.

Object persistence is done at runtime in O2, by linking an object to at least one *root of persistence* object. Objectivity needs to have persistence defined in the schema by deriving the classes from `ooObject` class.

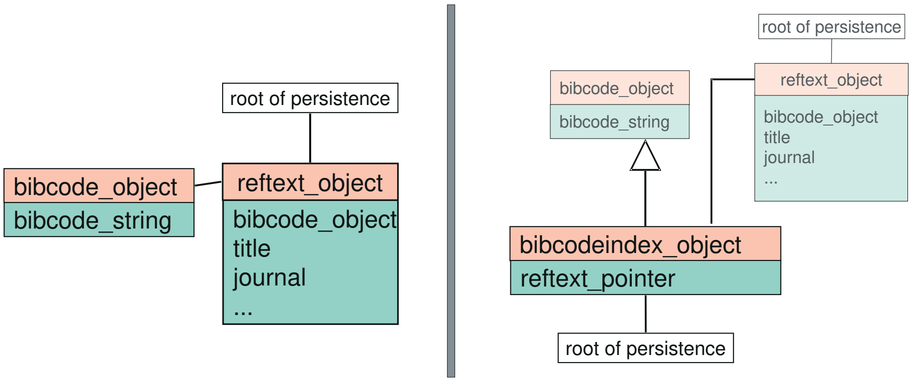

Figure 2. O2: Special class for indexing.

Objectivity/DB allows full control over clusterisation of objects through containers, whereas O2 allows only to give clusterisation hints, leaving the system free to manage them.

Locks are done at object level in O2 and container level in Objectivity/DB.

Indexing in O2 may only be performed on single fields, in objects behind a root of persistence. On the contrary, Objectivity/DB allows building an index on multiple fields and crossing aggregated objects.

4. Design and Implementation Problems

Differences in the behavior of two ODBMS may lead to different database design and implementation to overcome feasibility problems and performances.

This shows up for instance in indexing. The two following examples will show how the data model and the implementation have to be adapted to fit particular requirements of each ODBMS.

4.1. O2: Special Class needed for Indexing

In the left schema of Figure 2, the `reftext_object` contains the bibcode_object as an embedded (aggregated) object. Indexing on this last object is required to be able to access directly a reference text from its bibcode. However, O2 does not allow an index to cross aggregated objects, thus impeaching to index on `bibcode_object` through `reftext_object`.

It is therefore required (schema on the right) to create a new object class, `bibcodeindex_object`, derived from the `bibcode_object` and to add to it an association with the `reftext_object` by the mean of the `reftext_pointer`.

This new class, having its own root of persistence, can now be indexed, because it contains, through its superclass, `the bibcode_string`.

4.2. Objectivity/DB: Special Load Strategy

When loading a database, the following algorithm is usual when new data are to be entered according to some key:

1. If the object corresponding to the key already exists, just add the data;

2. If not, first create the object, then add the data.

In such a situation, it is useful to have an index and to keep it up-to-date, especially when the collection is large.

Objectivity is not efficient enough when using and updating an index at the same time. So a special loading strategy is needed for this purpose: (1) extract all keys and build a list of unique keys; (2) check for existence of the keys in the database, update the already existing objects and prepare a list of the new keys; (3) add the new objects; and finally (4) rebuild the whole index.

This strategy generally requires two readings of the data, but proves to be much more performant than the regular way of updating with Objectivity.

5. Results

Having in mind that the present SIMBAD software is a dedicated software running since a decade, the ODBMS do quite well:

Roughly speaking, Objectivity is 20% to 50% slower than the SIMBAD dedicated software, taking into account that the prototype did no binary encoding of the data; O2 is slower than Objectivity by a factor of 2. This is probably the price to pay for a system which is more dynamic at run time than Objectivity. It may be a problem for very large data collections (over 10^8 objects), but allows on shorter ones much more powerful applications.

Finally, both systems need much more space on mass storage: about 2.5 GB, where SIMBAD needs about 800 MB.

6. Conclusion

If learning an ODBMS system is easy, developing a performant application is not that easy. It requires to study carefully the precise characteristics of the ODBMS and to gain some unwritten experience with it, in order to optimize the implementation. Today, design and implementation of an Object Oriented Database cannot be independent of the chosen system.

In spite of these discrepancies, ODBMS prove to be mature database tools. Based on objects rather than on well structured tables like in RDBMS, they are well fitted for two kind of applications: those manipulating heterogeneous and complex data, and those dealing with huge collections, billions of objects and TB, where RDBMS give up.

References

Wenger M. et al. 2000, A&AS, to be published