

BAROUDI BLOOR

*The Failure of Relational Database,
The Rise of Object Technology
and the Need for the Hybrid Database*

“Object technology brought programming languages closer to natural language and quickly came to dominate thinking in most areas of software development.”

“Today’s developers use OO features as a matter of course.”

A Revolutionary Paradigm

When first introduced in the late 1960s, object technology was revolutionary. By the late 1980s it was becoming mainstream, for good reasons. Not only did it simplify interface development, it provided a more flexible and capable way to deal with data that fundamentally changed the way applications are built. Instead of representing data in rigid tables the way relational databases do, object technology describes data in terms of classes. An object is an instance of a class in the same way that a specific oak tree is an instance of the class of “oak trees”.

Object technology contributed the concept of inheritance, allowing classes to be arranged hierarchically. The class of “oak trees” could inherit data structures and behaviors from the more general class of “trees”.

Object technology maps well to the way we see the world, and OO languages proved to be more versatile in most areas of programming. They brought programming languages closer to natural language and dominated thinking in most areas of software development. OO was referred to as “the new paradigm” and its influence stretched far and wide.

OO features were quickly added to many established languages, giving rise to such languages as C++. New OO development environments appeared, including Visual Basic, Visual C++, PowerBuilder, Delphi, and Caché. Although championed in advanced development environments, it took some time for object technology to find its way into formal curricula. It’s taken even longer to conceive of and build a true object-based world – and we’re not yet there.

Spreading Object Technology Around with the World Wide Web

As the World Wide Web transformed information exchange of all kinds, the OO programming language Java became the darling of Web developers. Based on C++, Java could be used to create small applications (Java applets) that could be used from a browser.

Sun provided the Java environment free to promote its uptake. Within a few years, millions of copies had been downloaded and Java was everywhere. And Java gave rise to more OO languages such as JavaScript, C# and JScript. Internet development spawned other new OO languages like Perl and PHP. Today’s developers use OO features as a matter of course.

The Rise of Objects

Object technology informs almost every aspect of software development. OO modeling dominates the application modeling market with standard UML methodology taking the lead.

The 1990s saw the emergence of OO middleware products created to provide secure communication services between OO applications. The middleware market took a significant leap forward with the introduction of Java Messaging Service (JMS) in 1998. JMS defined a comprehensive set of messaging APIs and mandated that certified J2EE implementations had to include a JMS server. This enforced standardization, bringing down the cost of middleware and providing a platform for writing enterprise-wide object-based applications.

XML and Web Services

In 1998, HTML, the mark-up language used to specify webpage layout, was extended and standardized to create XML (eXtended Mark-up Language). XML provides a grammar that can be used to create self-describing data formats, similar to data definitions held in databases. With XML, programs can attach definitions to data and exchange both data and the meaning of the data. XML enables the definition of specific standardized data objects such as an invoice or a purchase order to facilitate data exchange within a company or between companies. XML gave rise to Web Services – programs able to interact with other programs on the fly without the need for customization. Now two formal Web Services environments, J2EE and .NET, have emerged. Today XML has a powerful impact on data. Like SQL, XML provides a standard for developers to get at data, but XML also provides a standard language for defining data at the level of the object. The growth in popularity of XML has been almost as dramatic as that of OO. As a consequence, new standards for data objects and new development products based on XML keep emerging.

Object Databases – The Missing Link

The rapid adoption of object technology in almost every dimension of software development stands in stark contrast to the slow, and until recently, limited adoption of object databases. The reasons for this sluggish uptake are many.

Early OO languages gave no thought to data storage. Programs worked on data in memory and stored the whole data image as a file to be read when the program was next run. This approach made it impossible for applications to share data and made data recoverability, manageability, and scalability impossible as well.

Not surprisingly, a variety of object database products emerged on the market: Versant, Objectivity, ObjectStore, GemStone and others, to provide appropriate data stores for OO development environments. These products were met with a good deal of initial enthusiasm and with the general expectation that such products would be able to carve out a significant piece of the database market – perhaps even dominate.

Unfortunately, by the time object databases were introduced, relational database vendors had already gained momentum and market penetration. It was easy to write OO routines to access relational databases with their standards-based SQL interfaces. In contrast, most early object databases provided no SQL capability at all and were not suitable for query applications. As a result, object databases never established a strong foothold in business systems. They did establish niche markets in application areas that manage and store complex objects such as CAD/CAM, telecommunications, multimedia, artificial intelligence, modeling financial instruments, patient care tracking systems and scientific applications.

The database market never paid a lot of attention to object databases until the emergence of XML as a data definition language, which prompted their reemergence to manage XML defined data, for which they were naturally suited. The use of XML, in combination with the escalating need to store complex data, seems to be leading the general resurgence in object database.

A survey of developers published by InfoWorld in September '03 shows surprising results. While 89.2 percent of respondents said they use relational databases, 52 percent of respondents also say they use object-oriented or XML databases. In answer to a question about the types of data stored, 40.2 percent said they stored persistent objects, 58.9 percent XML data and 89 percent relational data. Baroudi Bloor believes Object databases are more widely used than is generally suspected, having gained market penetration quietly in response to a pressing need.

The InfoWorld survey also clearly indicates that OO languages are the dominant choice for new development. We believe these statistics reflect a quandary developers face today. They need databases that work well with the OO languages they're using but they still need the query capabilities that relational databases provide.

Relational Databases – How the Other Half Lives

As soon as there were programs, there was data. One of the earliest drivers of the business value represented by IT was data management. Automated data management means business can scale and compete in ways that it can't without it. So it's no wonder savvy business technologists targeted the data management market early. More than a decade before object databases were even thought of, the relational theory of data proposed by Dr. E. F. Codd found its way into commercial relational database products. In the mid-80's an almost religious conviction in some parts of the IT industry held that all the theoretical problems of data had now been solved and that soon all practical problems would be solved as well. Obviously, it wasn't the case.

Relational databases represented data as being held in simple two-dimensional tables - an effective way to represent a lot of data in a way that programmers could easily understand. With SQL, relational databases established a standard data access language. They had a logical and physical structure that was application neutral and that worked well for many business applications.

However, one of the foundations of relational theory was the idea that data and the programs that use it could and should be independent of each other. This was, and is, at odds with the whole idea of object technology. Object technology encourages the designer to think of data as objects, not as tables. And objects and the methods that use the objects are not independent of each other.

Consider the automobile as a complex object. When you use an automobile, you use it in its entirety, as one thing – as an object. Associated with the automobile are many activities (methods in OO terminology). You steer a car, you change gear, you signal, and you turn the headlights on and so on. If the car is an object, these activities are the object's methods and they are fundamental to the car. The idea of these activities being independent of the car is ridiculous. When you put your car in a garage, you store it as one thing complete with its capabilities. You don't put your car in the garage and store its steering, transmission, signaling and lighting functions someplace else. Data and its associated processes cannot and should not be separated, and in object databases they are not.

In reality, both viewpoints have merit and limitations. Some processes are genuinely independent of the data. This is particularly true of queries that access large sets of data. A query simply selects some data according to a set of criteria and does not care about what the data is or how it is organized as long as it can be retrieved quickly. Queries are data-independent, but an object's methods are not.

Relational Database Limitations

Relational databases are more constrained in their capabilities than most would suspect. Storing and representing some fairly common data structures can be very difficult. Consider a bus route – a simple, ordered list of bus stops. Relational databases only hold tables as unordered lists and can retrieve an ordered list only if a specially built index is added. An object database has no problem with an ordered list and needs no index – the index being an artificial creation that exists only because of the limits of relational data structures.

Another common example is a bill of materials – a product and its components in a manufacturing system. The components themselves may have components that in turn may have components and so on. A relational database table of all parts will not express the relationships of the parts to the parts of parts, and so on. These relationships express important data. To query a database for a product and all its components should be straightforward. A relational database structure makes the developer's job of answering this simple query, unnecessarily complex and difficult.

Examples of this kind abound: A map and its roads, rivers, and landmarks; A web site and all its pages, links and graphics. In fact, the more complex the collection of information is, the more levels of hierarchy and cross relationships, the less possible it is to represent it within the simple table structures of a relational database. Object databases have no such limitation; indeed, they were designed to address this very kind of problem.

Despite the maturity of relational database products and the dramatic growth in computer power over the past decade, we still hear about projects that fail because the performance of the relational database used is just not good enough. Usually this is because of the way relational databases physically store data. For developers to assemble the data that they need, they often have to do multiple JOINS of one table to another to another to another. To retrieve the data, the database runs optimization routines to determine the best way to gather the data and then retrieves it. This process often takes a long time and can negatively impact performance. While relational database optimizers have improved over time, they still present performance overhead that is significantly greater than object databases.

Relational Databases and the Impedance Mismatch

The problem with relational databases is that the fundamental data structure they use is a two-dimensional table. In relational theory, data is supposed to be organized into normalized tables – that is, the data is supposed to be organized in such a way that there is only one way to get to it, allowing the developer to eliminate redundancy and ensure that changes to the data are consistent. This design technique was introduced to ensure that relational tables contained independent sets of data that were related only by a key. It derived from the mathematics of Set Theory, but the problem is that Set Theory is not capable of representing all the relationships and structures that data can have.

“...we still hear about projects that fail because the performance of the relational database used is just not good enough.”

“Indeed we have seen estimates that OO developers using relational databases spend anywhere between 25 percent and 40 percent of their time writing code to map objects to relational tables.”

Storing data in a normalized form often demands that the programmer disassemble an object, prior to storing it in the database, and reconstruct it, using SQL requests (multiple JOINS), in order to use it. It is as if, in storing a car in the garage, you take the doors off, remove the seats, take off the wheels and so on. It is time consuming and it makes no sense.

This problem surfaced when OO languages rose to dominance, and is usually described as *the object-relational impedance mismatch* connoting the difference between the approaches to data used by OO languages and relational databases and the resulting problems imposed on the programmer to address it. In reality, most relational databases are not completely normalized when they are implemented, but even so, impedance mismatch problems occur, making life difficult for the developer. Indeed we have seen estimates that OO developers using relational databases spend anywhere between 25 percent and 40 percent of their time writing code to map objects to relational tables.

Perhaps this fundamental difficulty should have created a strong demand for object databases, but there was a big problem with most object databases too: They provided little if any support for SQL. Many software tools require an SQL interface. In particular, business intelligence applications. Even object databases that had SQL interfaces were not built for managing the kind of query traffic that business intelligence applications generate.

The Object-Relational Database

Relational database vendors did not ignore the advent of objects. It was quite clear that normalizing complex data items made no sense. To take an extreme example, if you normalized a bitmapped image – which is an ordered list of pixels – you end up with a table whose rows are pixels and their attributes with a primary key that reflects their order. It is obviously better to store the data as an object.

They proposed the idea of the “Object-Relational” database. The idea was to preserve the overriding relational database structure, but allow columns in a relational table to contain complex objects. These objects could be accompanied by processes (stored procedures of a kind) that could be used to unravel the complex data. SQL was then enhanced to allow calls to the relational equivalent of “object methods”.

This approach mocks the relational theory of data, in fact it ignores it completely, but it does allow complex objects (maps, vector graphics, photographs, even whole tables) to be defined within a relational structure and held as items. So these capabilities were implemented and even branded. Informix referred to its embedded processes as DataBlades, while Oracle referred to them as Cartridges.

This provided an alternative to object databases for storing objects, but the fundamental problem had not been solved. Object-Relational databases still suffer from the impedance mismatch problem.

Object Databases versus Relational Databases

In practice, object databases have significant advantages over relational databases. Typically,

- They run much faster for transactional applications
- They handle complex objects far more effectively
- They offer better developer productivity
- They are easier to manage

In some cases, object databases have replaced relational databases for performance reasons. This has even been the case in large-scale business applications that did not involve the storing of complex objects – what some might think of as the natural domain of relational database.

The major performance advantage that object databases have is that they don't usually have to assemble the data before it can be used the way relational databases do. They tend to store data in its most-used form, which typically helps performance. Object databases can implement caching strategies that make it more likely that data is in memory when it is requested. They require little optimization to retrieve data.

As new systems are developed, the need to manipulate complex data such as documents, sophisticated graphics, web pages, multimedia, etc. continues to increase, and these kinds of demands are better served by object databases.

The OO Landscape Today

We've seen persistent growth in the adoption of object technology in every area of software development. Even in the last area to hold out – database – although object databases per se haven't displaced relational databases, InfoWorld reports that 52 percent of developers are using object-oriented databases or XML databases which are often object databases themselves. And some are choosing some sort of hybrid form that enables easy use of OO constructs. With the Web interface being a strong choice in new applications development, and Web services the targeted mechanism for application interaction, building for and in an object-oriented world seems to be today's reality.

The September '03 InfoWorld survey also shows that programmers are almost ubiquitously using OO languages. Indeed, though some claim to be using straight C, OO languages appear to be the language of choice for 90 percent of today's programmers. The survey also proclaims programmers' strong preference for Web-based apps and easy-to-use object programming and scripting languages. As more and more software engineers enter the market with formal OO training, object-oriented technology will be the only respected basis for new development.

“This hybrid opens up the possibility of having a single database engine, with a single set of data definitions appropriate for all applications.”

Conclusion

It may be that relational databases will continue to dominate the database market with object databases remaining in a niche market. Or object databases may enjoy increased market share, as they are better able to deal with the complex objects in use today. However, we suggest another possibility: Database technology could evolve to produce true hybrid products that offer the virtues of both a relational interface and object interface. We know this is possible. Indeed, at least one product, Caché from InterSystems, already does this. (The Caché database, incidentally, describes itself as neither a relational nor an object database but as *post-relational*). Database vendors – no matter whether their products fall into the relational or object category – could move in this direction.

The approach involves providing the database with a mapping layer, through which developers access the database. The mapping layer should be based on open standards to resolve the impedance mismatch problem.

Database calls could then be made either in SQL or as direct requests to an object class or collection of classes. The mapping layer would translate these calls into physical data requests to the database to retrieve the data. This would obviate the impedance mismatch.

Changing either type of database product to work this way will be challenging. Object databases would need fast indexing capabilities for retrieving queries selected from large sets of data. The relational databases that do this best use bit-mapped indexes but these often create overhead when data is updated. Because of this, few object databases have this capability.

Relational databases for their part would have to provide much more flexible physical data structures. In their evolution they have tended to implement tables at a physical level. They would have to abandon this inflexible constraint and allow the storing of variable data structures. The benefits to database users would be enormous. Consider uniting the advantages of object databases combined with the advantages of relational databases:

- Good transactional performance
- Complex data management
- Ease of management
- Rapid development
- Flexible query capability
- Standard data access interface
- Suitability for business intelligence applications

This hybrid opens up the possibility of having a single database engine, with a single set of data definitions appropriate for all applications. Baroudi Bloor believes that we, the Industry, need the hybrid database. Vendors must shove aside their religious predisposition and embrace a hybrid approach or be left to molder with COBOL and punched cards.

Copyright 2003, 2004 Baroudi Bloor International, Inc.

This paper was written by Robin Bloor of Baroudi Bloor, a research, analysis and strategic advisory company serving the IT industry.

Robin Bloor is Research Director of Baroudi Bloor International Inc and President of Bloor Research., one of the world's leading IT analyst and consultancy organization distributing research and analysis to IT user and vendor organizations throughout the world. Contact him at robin@baroudi.com.



BAROUDI BLOOR

175 Pleasant Street ◀ Arlington, MA 02476 ▶ 617-747-4045 ▶ www.baroudi.com