HURWITZGROUP

Fresh Thinking on Databases For Object Development

— Fresher Information Corporation

Hurwitz Report

HURWITZGROUP

Fresh Thinking on Databases For Object Development

— Fresher Information Corporation

A Hurwitz Group white paper written for:

Fresher Information Corporation
575 Market Street, 13th Floor
San Francisco, CA 94105
Tel: 415 356 8100
Fax: 415 357 4841
www.fresher.com

Published by:
Hurwitz Group, Inc.
111 Speen Street, Framingham, MA 01701 ▶ Telephone: 508 872 3344 ▶ Fax: 508 872 3355
Email: info@hurwitz.com ▶ Web: www.hurwitz.com

January 2002

Executive Summary

If corporate developers have shifted to using object-oriented techniques, why not corporate database administrators (DBAs)? The unique benefits promised by object databases in the mid-1990s, as the natural database twin to object development, never disappeared. The benefits did get clouded over somewhat by the slow rate at which corporate developers adopted object-oriented techniques, plus the lack of object databases' support of SQL, which remains to this day the *lingua franca* of structured data.

A new day may have dawned for object databases however, for object development now rates as the norm, not the exception, in corporate IT shops, and a few object database vendors have overcome some of the limitations which prevented the pervasive use of object databases in the mid-1990s. Hurwitz Group takes a look at Fresher and its database Matisse, in this report, and considers Matisse's prospects given the current market.

The report first looks into recent history, in terms of the adoption of object development, and the possibility of renewed interest, therefore, in object databases. It also examines why object databases didn't quite become the market force many predicted for object databases in the mid-1990s. Next it examines what an object database vendor might need to offer in order to ride the current and future popularity of object development, in order to create an object database renaissance, and finally looks into how Fresher's Matisse addresses these criteria.

Finally, the Era of Objects Has Arrived

It started in the late 1980s and promised to take over the world of software during the 1990s — of course we are referring to objects. It didn't quite happen as quickly as many pundits thought, however. By the mid-1990s, rapid application development ("RAD") tools, like Visual Basic, Powerbuilder, and Delphi, had emerged as the programming market's darlings. Strictly speaking, these tools were at best "object-based," and did not directly employ object-oriented ("OO") methodology (you could do it with Delphi, but many developers didn't approach it that way). Also by the mid-1990s, C++ had largely replaced C, but in name only. Many developers were using C++ compilers, but really still writing C, only using object-oriented principles where they were strictly enforced — and C++ in most cases offered an opt-in OO model.

Yet today, in early 2002, the era of objects has finally arrived. From a numeric perspective, a glance at Figure 1 illustrates how object-oriented programming languages, in total, will exceed usage of all other programming languages in 2002; and some would argue that the current version of Visual Basic now uses a true OO development metaphor. What happened over the past five years to make OO so prevalent?

- In latter 1995, a new, strictly OO language came onto the market, Java, which has rapidly grown in popularity.
- Many of the C developers using C++ tools have switched to using the true OO aspects of C++.
- Most universities changed their computer science curriculums during the 1990s to teach OO methods.
- Companies began to get OO religion; the promise of improved productivity and quality convinced many corporate IT departments to adopt OO, at least partially.

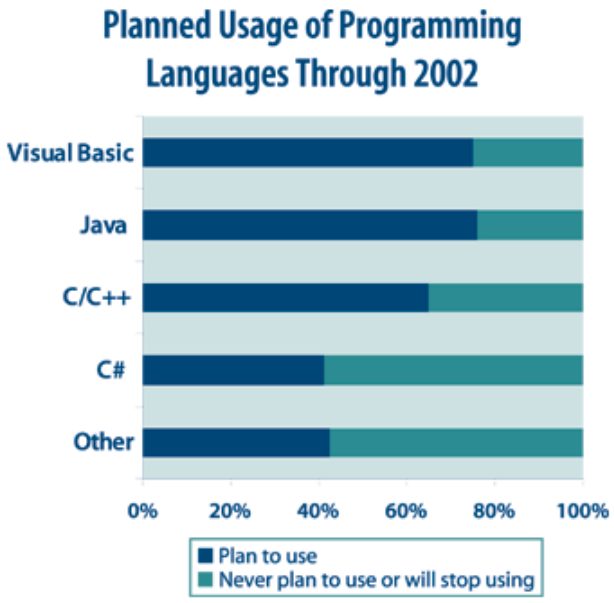

Figure 1. Planned usage of programming languages.

Source: Hurwitz Group IT Decision-Makers Study, 2001, N=180, North America

Thus, in today's programming market, OO tools, and corresponding OO-based server architectures like Java 2 Enterprise Edition (J2EE) and Microsoft's .NET, own the market majority. The recent hype wave, which will slowly but certainly turn into a commercial wave, about XML and Web Services will further propagate object-oriented computing.

Development and Database: A Marriage Made in OO Heaven?

Let's go back to the mid-1990s again. Since objects were supposedly revolutionizing software development, couldn't they also offer a more modern alternative to the middle-aged relational database? Several forward thinking software vendors thought so, and several object databases were brought to market during this same mid-1990s timeframe.

Object databases promised the same value proposition as OO development, with design accuracy and solution flexibility the leading benefits. For example, with OO development object designers are able to emulate solutions using the native language of the solution, rather than the language of the computer. Pretend you were designing a software solution to run an elevator; with OO methodology you define the software pieces, the object "types," by describing the actual elements of an elevator system (the pulleys, the cables, the elevators doors, the human user interface, a.k.a. the "buttons," etc.). You could also describe the actions of ("methods") and the interrelationships between ("interfaces") these objects in a direct manner, such as PressStopButton or ReleasePulleyWeight.

In addition, OO makes it easy for developers to change solutions to deal with real world changes. For example, "We've decided to build a 10 story building instead of eight stories!" would result in simple underlying code changes rather than design changes. "We've decided to add another elevator!" could be handled with just another instantiation of the elevator object class. OO also offered reuse — the elevator object class, and all its related objects, could easily be reused and repurposed to run an entirely new elevator system.

Similarly, the object database persists the data, information, content, and/or object itself associated with ReleasePulleyWeight using the tongue of the solution rather than the computer. Rather than force fitting all information into a handful of predefined data types, like integer, long, or text (of a certain length), object databases allow developers and database administrators to define and use data types that directly reflect the real solution — object types. The object database handles the dirty task of figuring out how to actually physically configure and store information — this physical information design and implementation abstraction empowers database experts to remain focused on solutions.

The fact that developers and database designers could work together using the same solution metaphor made for an enticing productivity story back in the mid-1990s. The fact that object-based application runtimes could talk to object-based data management runtimes promised unheard of levels of solution accuracy, maintenance flexibility, and performance.

Early to Market — The Object Database

Despite all the early promise, history has shown that objects did not change the world of software development, and thus of database engineering, overnight. Four primary inhibitors prevented object databases from achieving greater market penetration during the 1990s:

- **Not enough object developers.** Though academic and many software vendor developers quickly embraced OO in the mid-1990s, the majority of designers, developers, and database administrators (DBAs) in corporate IT were too heads down in work to take the time to learn about object-oriented techniques. Eventually, over a period of five years, corporate developers and database administrators learned about the power of objects. In the mid-1990s, however, there were simply not enough object developers in corporate IT who could grasp the value proposition of object databases.

- **Performance.** The object database offloads some of the everyday aspects of database design and administration by automatically deriving physical database implementations from object representations. Particularly during the first generation of object databases, haphazard object design and/or management by developers could result in performance penalties in the resulting object databases. In those early, experimental OO days, too often developers persisted their objects indiscriminately (using a technique sometimes referred to as transparent persistence). They did not grasp that haphazard object designs and loose object persistence could directly result in raising the price of achieving desirable performance in the object database.
- **Reliability and availability.** By the mid-1990s most relational database vendors had developed a bevy of management tools to enable DBAs to tune performance, manage backups, automate failover, and in general make relational databases work with a high degree of reliability. Not all of the relatively youthful object database vendors invested enough in management tools to ensure high reliability and availability.
- **SQL.** Few object databases supported SQL in the mid-1990s. SQL, the *lingua franca* of structured data, was and is supported by all of the relational database vendors. Relational database vendors were not enamored with the idea of losing market share to the object database upstarts. The lack of SQL support by object databases gave relational vendors a key advantage in protecting their shares. At the same time, the relational vendors need to respond in some fashion to the object database challenge, so they invented a hybrid "object-relational" or O-R database. Though the O-R database's support of objects involves force-fitting objects into a limited physical typing metaphor required by their relational core design, O-R databases support SQL. Given the commercial environment of the mid-1990s, the total lack of SQL by most object databases, vs. the partial support of objects by adapted relational databases, kept most buyers in the O-R camp. No one in the mid-1990s offered the best of both worlds: pure object support plus complete SQL support.

Now that object development ranks as the norm rather than the exception, has the time come for an object database renaissance? After all, the object database's original value premise remains utterly valid.

An Object Database Renaissance?

Driven to a large degree by the interest and popularity of object-oriented server technologies like J2EE and Microsoft's .NET, certainly conditions exist for object databases to make a compelling comeback. Paradoxically, some of the reasons that object databases were not chosen in the mid-1990s now will entice buyers to object databases. Specifically:

- **Performance.** J2EE and .NET are pure object-oriented environments. The objects and object containers (such as Enterprise JavaBeans, EJBs, in J2EE) go through a translation known as "mapping" to work directly with SQL-based relational databases. The extra code associated with the mapping yields a small performance penalty in each occurrence, but in large, transactional systems, this object-relational mapping can actually drag performance down by as much as 50%. Much of .NET remains in beta, but J2EE-based applications, particularly those using J2EE's more sophisticated features often handled through application servers, have had to deal with a less than sterling performance record. Object databases offer a natural performance benefit for such pure OO, transactional applications because they eliminate the need for mapping.
- **Flexibility.** O-R databases offer full SQL compliance, but limited object capabilities; the actual database engine remains optimized to process SQL and relational designs. Some modern object databases that also support SQL, however, offer the best of both worlds: If you need full-blown support for objects including classes, inheritance, and relationships, these object databases offer the widest functionality and best performance; if you also need SQL support, these object databases can easily support protocols like JDBC and ODBC, and even native language and platform support in some cases.

Remember, SQL/relational database processing is a simple case of object database processing. O-R databases must jury-rig their object support into a suboptimized (for objects) SQL/relational framework. Hurwitz Group sees a number of projects requiring both strongly-typed object (and data) development, plus a little SQL. How do buyers handle this condition? Object databases that support SQL offer the best total database flexibility to deal with such challenges.

- **Integration.** Larger information technology projects usually require more project time for integration than writing new code or building new data stores. Many integration-intensive solutions require their own data stores, either as part of the actual integration requirements (such as joining object-oriented computing resources with SQL resources), or for persisting value added information resulting from the integration processes. Object

databases, particularly those that also support SQL, make natural companions for such integration intensive projects. Object databases are the perfect candidate if you need:

- Bridging between object and nonobject resources
- Support of several language bindings (like Java, C, and C++)
- An object-oriented integration solution, including the support of OO design through standards like UML (unified modeling language)
- Standards-based support for both data representation from internal applications (SQL) plus external integration (XML)

One object database vendor in particular maintains its natural object database benefits for pure OO applications, but also fully supports SQL to handle migrated code and/or expertise limitations, plus XML for external and future integration.

Fresher's Matisse Bridges the Object-SQL Gap

A San Francisco-based private company founded in December, 1998, Fresher acquired ADB, Inc., of France, and thus the Matisse database, immediately upon Fresher's corporate launch. Matisse has been battled tested, particularly in Europe, for over a decade in huge, mission-critical transactional applications. Early on, Matisse differentiated itself from other object databases by excelling at performance and reliability — in fact several nuclear power plants in France run on Matisse. Matisse's original design objectives included the ability to run in a real-time, no downtime, mission-critical environment

Given this heritage, Matisse is a natural for high availability process manufacturing and other forms of complex real-time applications solutions — particularly those built with C++, Java, C# or other object oriented languages. Other decision-focused applications, such as broker/dealer trading systems, high volume and high complexity supply chain management, telecommunications, and real-time medical systems also benefit from Matisse's object, SQL, performance, and reliability story. In addition, multimedia, given its unique indexing and use case requirements, should fit nicely with Matisse's value proposition.

Since the acquisition of the Matisse technology, Fresher has taken several significant steps to optimize Matisse even further to handle the rigors of object-oriented, real-time, high reliability and performance solutions. The recently released version 5 of Matisse exhibits some of the following key features and benefits:

- **SQL.** Matisse fully supports the SQL 2 standard, including stored procedures, triggers, and object extensions, enabling Matisse to be used in dual mode by object and other types of developers. Matisse also supplies referential integrity, an absolute must for relational types of database designs. Matisse's design also eliminates the typically clunky handling of complex joins, meaning that Matisse's SQL may actually out-perform relational databases with complex table interrelationships.
- **Performance.** Matisse excels in terms of high scalability, mainly due to its optimized use of kernel threads, which yield linear or near-linear performance on symmetric multiprocessing systems (which support multiple, dynamically allocated CPUs). Matisse also includes sophisticated caching options, plus an abstracted versioning architecture to optimize hardware-specific performance. In addition, Matisse's versioning engine design ensures data consistency even in the most dynamic transactional environments, meaning it can beat relational databases in performance in high usage environments.
- **Reliability and administration.** Matisse possess all the most modern types of administration utilities, including automated parallel backups (that is, without administrator intervention and without operational interruption), on-going disk optimization through dynamic load balancing, and automated caching and space allocations. It also supports disk mirroring and replication for five 9s reliability. In addition, its versioning engine guarantees object viability by maintaining all references, even if an object has been updated real-time.
- **Productivity.** Of course, Matisse handles objects directly, making it a natural for applications built using J2EE, .NET, and other object solutions. Matisse, however, also supports a variety of languages in a language independent fashion, meaning, for example, that object applications developed in a combination of Java and C++ can share the same database objects. XML document schemas map perfectly to Matisse object structures, and Matisse offers an object API to directly handle XML documents. Matisse also supports automatic loading and mapping of XML documents and schema in response to SQL queries. Finally, Matisse fully supports high-end media types, including streaming media, and text indexing for optimized search.

Given the types of advantages Matisse 5.0 offers, Figure 2 illustrates the types of solutions where organizations might consider deploying Matisse.

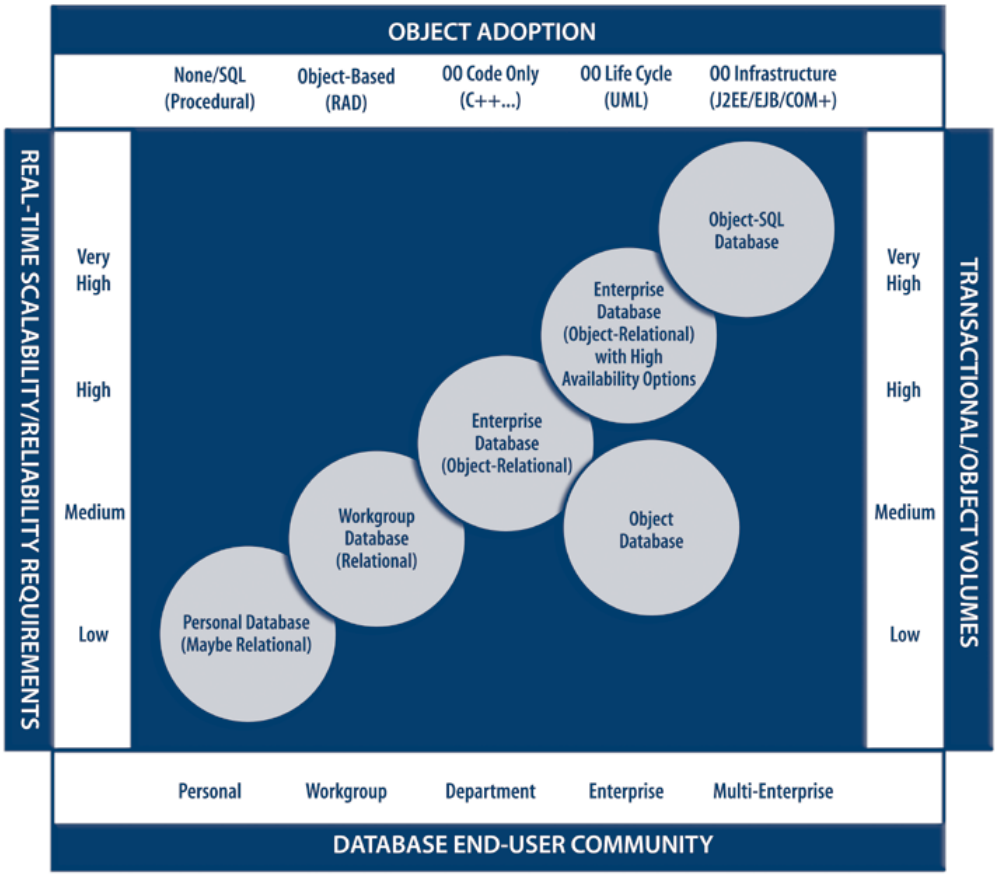

Figure 2. Database types for solution requirements.

Conclusion

Matisse Is the Natural Match for Object Development

Objects have reached the mainstream. In 2001, virtually the entire Global 1000 uses Java to some extent, many to a great extent, and the use of other object-oriented development architectures, such as .NET, and Web Services loom in the near future. For those organizations that take great care to invest in objected-oriented techniques for building and maintaining critical solutions, Hurwitz Group suggests you should also take great care in your choice of database. Certainly for applications making extensive use of objects, particularly in a high volume transaction or rich media usage environment, the relational or O-R database may not be the best choice. A high performance, high reliability object database, that eliminates the object-relational mapping penalty, that also does an excellent job supporting SQL and delivers the bridge to XML, could prove a far superior choice. Hurwitz Group believes that for many

solutions in many organizations, Matisse will pay off in terms of shorter time-to-market in the near-term due to shorter database design cycles, and lower TCO over the long-term due to lower costs required for database maintenance and performance.

Fresher can already point to a long list of references for its Matisse database, particularly in Europe. Fresher also can boast several key partnerships in Europe with systems integrators and software vendors. In order to engender success in the United States, Fresher need only replicate the types of partnerships it has already established in Europe; the reference list, already international in nature, speaks for itself. Certainly Matisse should look to work with integrators specializing in mission-critical development projects, and mainly those with strong J2EE-oriented practices, and those which have committed to enterprise-scale .NET. Matisse also extends the value proposition of J2EE application servers and development tools, so those J2EE vendors that most readily work with partners, such as BEA, SilverStream, Sun/iPlanet, and even Switzerland-like Sybase (even though Sybase has its own database) might yield synergistic partnerships. IONA and Borland, both leaders in terms of supplying object middleware, might also help round out, and be rounded-out by, Fresher. It is also imperative that all database vendors, whether an object, relational, hierarchical, or any other kind of heritage, keep their eyes focused on XML, for that is currently the highest wave in data management (albeit unstructured data management).

What makes J2EE so compelling is that it supports an organization's desire to use the latest in object-oriented development techniques, yet it does not require organizations to throw away existing assets. J2EE helps organizations achieve infrastructural and process modernization, while simultaneously improving ROA (return on assets — of an IT type). Fresher's Matisse promises, and delivers, on the same type of value proposition. As the most natural database fit for large-scale object-oriented solutions, yet also effectively supporting SQL and XML, Matisse extends the tactical and strategic benefits of object development.

About Hurwitz Group

Hurwitz Group, an analyst, research, and consulting firm, is a recognized leader in identifying and articulating the business value of technology. Known for its real-world experience, consultative style, and pragmatic approach, Hurwitz Group provides strategic guidance to its clients by delivering analysis, market research, custom content, and consulting services. Clients include Global 2000, software, services, systems, and investment companies.

111 Speen Street Framingham, MA 01701 T 508 872 3344 F 508 872 3355 www.hurwitz.com