db4o | The Open Source Object Database | Java and .NET

Enabling the Mobile Enterprise with db4o

By Eric Falsken

Eric Falsken is a long-time user of db4o in distributed medical devices and has recently joined the db4o team as Technical Evangelist.

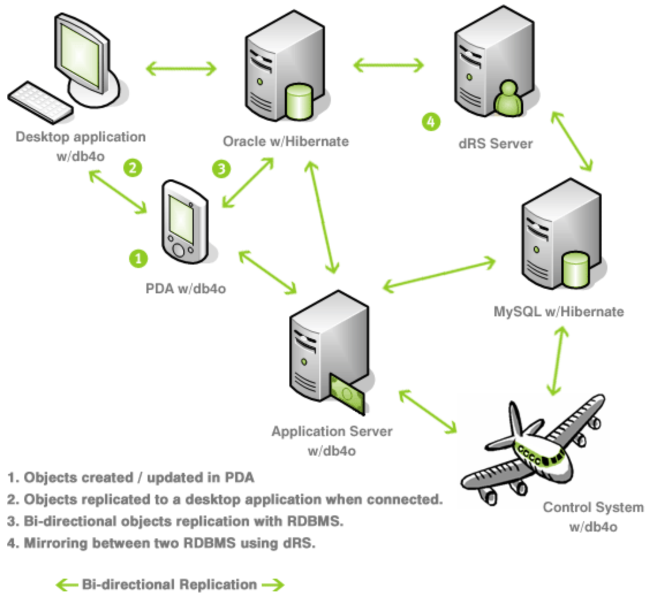
As an experienced enterprise developer and as an early adopter of Microsoft .NET (much to the chagrin of his open-source-loving co-workers) he enjoys exploring and developing tools to enable elegantly usable and maintainable software.

His blog is www.everylittlething.net

Emerging new business models and enterprise organizations break the boundaries of classic, networked computing and with it, the demand for more distributed applications and the mobilization of supporting data are reaching new levels.

Instead of filling out forms or spreadsheets for backoffice data processing, or wasting office time to enter data when back from a trip, the mobile worker now uses **portable client software** programs to enter or retrieve data while on the move - on a handheld, a smartphone, a tablet or a laptop PC. Also, **devices** are starting to processing their own data and synchronizing with backoffice enterprise systems to trigger business processes – for example a vending machine tracks local inventory implements on-demand restocking.

Businesses see huge cost benefits and faster execution times derived from the mobilization of enterprise processes and have started to update their corporate IT to facilitate the integration of this important source of competitive advantage into their infrastructure.


Desktop application w/db4o
Oracle w/Hibernate
dRS Server
PDA w/db4o
MySQL w/Hibernate
Application Server w/db4o
Control System w/db4o
1
2
3
4
1. Objects created / updated in PDA
2. Objects replicated to a desktop application when connected.
3. Bi-directional objects replication with RDBMS.
4. Mirroring between two RDBMS using dRS.
Bi-directional Replication

As a result of this trend towards mobilization, Java and .NET developers seek strategies to store data on distributed, client-centric devices and equipment, finding that classical, **relational database solutions fall short** when the environment demands zero-administration, small footprint and flexibility, e.g. the ability to update the data model of an in-production database which has been distributed to thousands of clients in the field.

While **50% of the developers currently still write their own persistence solution** to meet these requirements[1], a fast growing community of users now selects db4o's object-oriented database management system designed to meet exactly the needs described above.

[1] According to research by leading embedded software system analyst Chris Lanfear, director with Venture Development Corporation (VDC) in Natick, MA.

Companies including Hertz, Eastern Data, Mobilanten and Mandala have shipped their first mobile solutions with db4o, and many more are in the process of rolling out additional products.

db4o offers not only a mobile, native database for seamless client-centric persistence. Since January 2006, it also provides - by means of the db4o Replication System (dRS), powered by Hibernate - a solution to the problem of integrating distributed, object-oriented data with backoffice, server-centric, relational databases such as Oracle or MySQL.

In this paper I want to discuss mobile enterprise solutions in more detail by looking at a specific use-case to highlight the challenges and point the specific benefits from using db4o's mobile database for Java and .NET.

db4o Case Studies

- Mobile -

http://www.db4o.com/about/customers/platforms/mobile.aspx

Eastern Data's Mobile Apps

Mandala's Smartphone Software

1. The Case

The classical configuration for the mobile enterprise is a networked, server-centric data center with one or more centralized database instances and a large number of distributed, **periodically connected clients**. Periodically connected means: At some stage, you are offline, and hence need data to be persisted on the client's storage. But at some stages, you are online (wireline, wireless, or directly connected to a PC or a network), and thus able to sync data between the data center and the client device's storage.

Imagine a regional sales representative, Larry, who works for a company that makes widgets. Larry's Smartphone[2] is running a piece of db4o-enhanced software that can interface with his company's CRM, and order processing system.

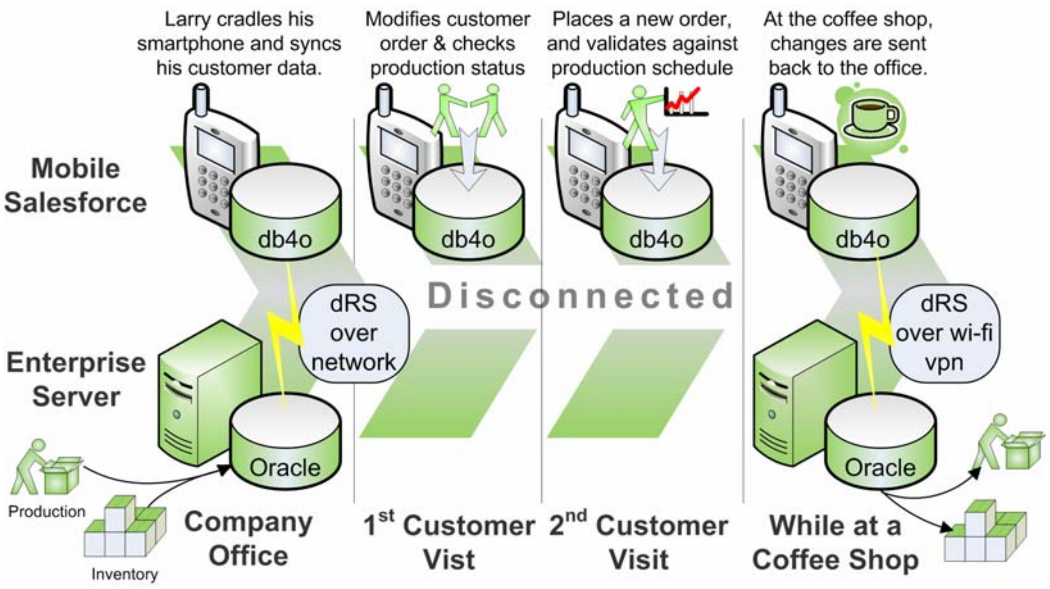
Larry is getting ready to leave the office to visit a few of the customers throughout his region. Because he knows that he may not get the best reception while on the road, he taps a button on the screen to synchronize his Smartphone, while still in the cradle next to his desktop PC, with the latest data on all his customers and their related (business) objects, such as orders.

Arriving at the first customer, Larry finds that he wants to postpone the delivery date of a previously placed order. Since Larry already has all of his customer's order information, Larry pulls up the order on his Smartphone and negotiates a new delivery date.

While discussing potential dates with the client, the software checks their validity against **business rules, which were stored together with the "order" object**. Those rules observe a variety of business factors, such as the company's production schedule optimization algorithms and certain basic inventory and key resource availability information. In fact, one of the business rules had just been updated to observe the availability of another

[2] db4o and the dRS support many runtime platforms. In addition to Windows Smartphones, db4o also supports .NET- and Java-powered devices. For a complete list of platforms, please see the product datasheet.

key input component, reflecting recent changes in the product specification. But due to the object-oriented nature of the data synchronization and the automatic schema evolution of db4o, the smartphone had stored this information without the need to do any administrative tasks, such as to create or amend a table in a relational database. Because of this ease of updating, Larry's company has found it much easier to update production schedules and thus to stay competitive in the marketplace.


Larry cradles his smartphone and syncs his customer data.
Modifies customer order & checks production status
Places a new order, and validates against production schedule
At the coffee shop, changes are sent back to the office.
Mobile Salesforce
db4o
db4o
db4o
db4o
Disconnected
dRS over network
dRS over wi-fi vpn
Enterprise Server
Oracle
Oracle
Production
Inventory
Company Office
1st Customer Vist
2nd Customer Visit
While at a Coffee Shop

On the second visit, the customer's order has just been delivered. The customer wants another hundred units, so Larry pulls up the customer's previous order and adds a new order for the same items.

When Larry stops at the coffee shop a little later, he turns on his Smartphone, and lets it connect to his office via GPRS. With the push of a button, his updated orders are synchronized with the application server. Two activity items get added to his task list: The order with the modified delivery date has already begun production, so the modified delivery date was unaccepted. He calls the customer to let them know to expect it. The second customer's new order has been approved, and now has an estimated shipping date. He also calls this second customer to discuss the delivery date.

The entire data synchronization process is actually taking place within Larry's application software. The embedded db4o replication system (dRS) is capable of automatically replicating any modified data objects. The entire replication process is also capable of including business logic many times more complicated than traditional RDBMS replication is capable of. For example, in this case, the production team had just begun assembling the first customer's order. Since production had put a freeze on the order, the synchronization was rejected. Previously, logic like this had to be coded by hand into a network application. However, by using db4o and the dRS, the only code that needs to be written is a simple conflict resolution handler that checks to see if the order is already in production. The dRS API is also capable of performing automated tasks, generating events, or prompting the user to make a decision.

Once the data has been synchronized with his application server back at the office, Larry’s changes are immediately visible to other users. The new order begins to be processed in the production department, and purchase orders for the required materials can be generated automatically.

All of this could have been possible with conventional technology, but it woudl comprise many dozens of relational tables which all would need to be referenced by hard-wired, static application code, consisting of many lines including non-native SQL strings that make refactoring even more cumbersome. All of this code would be time-consuming to keep maintained on a regular basis for a rapidly-evolving product. Everything would also have to be duplicated for each platform that the data would be replicated onto: once from the mobile device to the application server, and again from the application server to the sales team’s analysis server. Instead, it took the developers only a matter of minutes to add a few lines of code to their application to run the download.

2. Using db4o to Enable Distributed Applications

Most modern applications are written in an object-oriented language and most real-world data is easily expressed in object semantics. So it makes sense that developers should consider db4o.

In business applications, it is much more valuable to use a “Customer object” rather than a “row from the Customers table”. For example, a Customer object can contain logic to determine if or when a customer can order a certain product, and how much that product should cost. Traditionally, developers would have to issue a number of database queries and run through a number of methods to figure it out, and such logic is poorly suited to be packaged for reuse and distributed storage on mobile devices.

db4o allows your developers to spend time on making their application work instead of designing database schemas and writing persistence code. In fact, db4o has been proven to reduce the cost of developing your persistence layer by up to as much as 90%.

Distributing db4o-powered applications is also easy. db4o is designed to be embedded into software components, and to be completely invisible to the end user. Deploying db4o is as easy as including a single file in your application installation. There is no costly database server to install, configure, and maintain.

Moreover, db4o will automatically adapt to application upgrades. It assumes the absence of a database administrator, and allows the application to seamlessly switch from the old database to the new one, resulting in easier, smoother and faster deployments and lower support costs. db4o almost completely eliminates the cost of maintaining your database when releasing a new version of your application.

db4o allows you to develop applications using Java or Microsoft .NET for deployment on PDAs, Mobile Phones, Windows desktops, Linux, and Java Servers … and they can all intercommunicate seamlessly.

3. Collecting Your Distributed Data

db4o's unique object-oriented replication (dRS) functionality allows for easy synchronization of data between db4o databases, relational databases such as Oracle, SQL Server, MySQL, or any combination thereof. Relational databases are wrapped by the popular Hibernate object-relational mapper.

The dRS is a 100% object-oriented software module. As such, it is the perfect solution for Agile enterprises with rapidly-evolving software environments. Because the dRS is designed for embedding into your applications, it keeps all of your domain logic and business processes within your application code instead of scattered among the different systems you would commonly find in an enterprise server room. Each of those systems would typically require their own development efforts. Instead, the dRS allows you to integrate and focus your development efforts into a single application codebase. The dRS significantly reduces the time spent developing and testing your distributed data functionality, and lets you spend more time developing applications that improve your business.

Application-integrated conflict resolution is a feature which is unique to the dRS system. Traditional replication is focused on the structure of the database, and meant that replication dealt in row-level or column-level changes. The terms “row” and “column” don’t mean very much in most applications. Wouldn’t it be better to synchronize real “customers” and “orders”? The amount of code required to handle a single conflict between two databases is almost completely eliminated in most cases: usually it is as simple as a single if-statement.

When conflicts arise, traditional data replication systems ship a log of conflicts to a processor. That processor adds flags to the database which application developers would have to check periodically before processing them. If the replication system is application-integrated, then the application receives immediate notification of any conflicts. The conflicts can then be handled immediately by the application or user.

Maintainability is key in any enterprise development effort. When developers hear the world “replication”, they typically go and hide under their desk. This is because any changes to the data format need to be synchronized with all-encompassing changes to all data stores involved, the replication scripts, and the data access code of all of the applications involved which may use that data. Using db4o and dRS means that your application can seamlessly upgrade your database whenever your application’s object model changes.

4. Conclusion

The mobile enterprise is imperative for companies that want to use technology-enabled business processes to differentiate themselves from their competition and/or that need to contain their costs at all stages of their value chain.

New business models and processes need the right technical infrastructure. The mobile software stack looks different from the server-centric one – and the leaders for server-centric databases (Oracle, IBM and Microsoft) are laggards in the embedded software space with market shares between 2 and 5%[3].

A new breed of forward-thinking developers is using the benefits of object-oriented computing to build more-distributed mobile enterprise applications. They bring object-orientation to the data layer to allow for rapid application development (with the benefits of lower cost and faster time to market) and to facilitate more flexible, and hence agile, business processes in an ever-faster-changing business environment.

db4o is the only native object database specifically designed for client-centric, mobile persistence tasks in both, Java and .NET environments. The newly released dRS fosters that position by providing a ground-breaking new replication system, that allows applications to sync even the most complex object structures with legacy relational databases - with only one line of code.

[3] According to research by leading embedded software system analyst Chris Lanfear, director with Venture Development Corporation (VDC) in Natick, MA.