ODBMS.ORG User Report No. 28/08
Editor Roberto V. Zicari- ODBMS.ORG www.odbms.org
November 2008.

Category: **Industry**
Domain: **Software development and consulting**
User Name: **Peter Fallon**
Title: **Director**
Organization: **Castle Software Australia Pty Ltd, Australia**

Peter Fallon is the director of Castle Software Australia Pty Ltd, a one-man company providing custom software development and consulting. He has been developing and supporting PC database applications since 1986, ranging from small business systems to large corporate installations.
Peter has spoken on topics covering programming, project management, Jasmine and object oriented techniques at many conferences from CA-Technicon 1992 to Devcon Australia 2003. Peter is currently working on a number of C# development projects, and maintains a number of commercial Visual Basic, C# and Office/VBA based products for clients.

Q1. Please explain briefly what are your application domains and your role in the enterprise.

Because I'm an independent software company, there is no one domain I work in - I've built systems for management consultants (Business Process Re-engineering), HR systems, Financial Software, Medical databases, and pretty much everything in between. The critical thing is that all of my systems are turnkey, many are sold as commercial software by my clients, and all are MS-Windows based.

My role is a jack-of-all-trades, responsible for initial problem identification through to final system implementation/installation and maintenance. As a one-man-band, I'm generally limited in the initial size of projects I undertake, but I follow a number of products over their entire lifecycle, including one system that is currently about 160,000 lines of code that has been maintained and enhanced over the last 15 years.

Q2. When the data models used to persistently store data (whether file systems or database management systems) and the data models used to write programs against the data (C++, Smalltalk, Visual Basic, Java, C#) are different, this is referred to as the "impedance mismatch" problem. Do you have an "impedance mismatch" problem?

Yes. I believe you cannot avoid it as soon as you have to write data from memory to disk. Either the (relational) database mapping causes it, or you have to create the impedance for optimization reasons, e.g.: where speed of overall data processing on the backend is more important that the speed of loading/saving a single form/record.

Q3. What solution(s) do you use for storing and managing persistence objects? What experience do you have in using the various options available for persistence for new projects? What are the lessons learned in using such solution(s)?

Most of my solutions have been custom created on a project by project basis. The reason is that I'm often hampered by having to work with an existing data structure, an existing corporate standard/constraint, or an obsolete database engine that isn't supported by the newer O/R tools (commercial products have a shelf life of decades, and we have to be extremely conservative when it comes to changing data formats).

One of the fundamental flaws of off-the-shelf OO solutions is that they assume you are creating the application and data structures (classes/tables) from scratch. Primary key and OID constraints particularly cause problems along with the foibles of pre-existing data (e.g.: Nulls in unexpected places because of old business logic that may or may not still apply).

As such I've generally first worked out the persistence strategy that will work best on an application by application basis, and then crafted small code generators that write my classes from the pre-existing tables, and embed in any code required to work around impedance problems...etc. This might sound like a lot of work, but its effort that is only done once at the start of a project - and for a system that might be maintained for many years, this is a sound strategy. For the most part, I'm not concerned about getting it done fast as much as getting it done right first time. My clients are small to medium sized businesses, which would fail if their software didn't simply just work from the moment an install CD goes in the drive.

A code generator also gets around language specific problems nicely (in terms of the amount of code required to solve a problem). In one old Visual Basic 6.0 system, it allowed me to be highly productive and write very stable code, even with a very complicated object hierarchy - as the generated code simply worked...

The old rule of time spent in design is time well spent is completely true.

I've found C# to be particular nice as a language to work with, both in terms of day to day practicality, and meaty enough to solve the inevitable "everything you never wanted to know about XXX but were forced to learn" problem that comes along. That MS finally took a page out of Delphi and others and wrote the majority of their Framework class library in C# itself means they have thrashed the language pretty well.

Q4. Do you believe that Object Database systems are a suitable solution to the "object persistence" problem? If yes why? If not, why?

Suitable, yes. Likely to be used a lot - not really. I find the theory and practice of object persistence to be very different things as for the kinds of custom/commercial products I build, the database requirements are only one small piece of the overall product/design picture. It is extremely easy for issues such as existing database constraints/policies, pre existing data structures you can't change...etc to nobble your desire to start with the purist solution.

Object persistence using various mapping techniques should always be a starting point however - and with modern languages there really isn't any excuse to the fully OO at least within the application domain. Coding of screens, reports and general logic is far simpler and faster if you don't have to worry about the storage/persistence end of things.

Q5. What would you wish as new research/development in the area of Object Persistence in the next 12-24 months?

Toolkits that allow much better interaction between persistence strategies and existing database systems - which allow interactive (2-way) table/class design, including various database strategies from writing direct data handling code class by class, to a centralized persistence engine (e.g.: I built a simple reflection based system in C# in a few weeks), to automating production/maintenance of stored procedures that might handle the nuts and bolts of object persistence on an (SQL) Server Database system so you can hand a DBA everything THEY want without extra effort. In essence, I want to be able to make a schema change and have ALL the code, database change scripts and refactoring all taken care of at one go.

As part of that, I'd like to be able to design my (business logic) classes as ONE unit - and just show via attributes or some other compile time declaration which methods should be executed server side - and let the application engine worry about the complexities of distributing the

application across the client and Server. This would include surfacing any existing server side stored procedures...etc and allowing me to simply include them directly as class methods...

Of course, these would be template based - so you aren't constrained you any one approach, and can tailor the code produced for your own purposes and libraries.

Existing products are either so ignorant of real-world conditions its painful (e.g.: scaling some Microsoft solutions), or assume you can start your systems design from scratch.
At best, tools exist to help you do each of the discrete tasks, but you constantly fall foul of forgetting a step somewhere and have to go back to the beginning each time...

And, my apologies to the Java fans out there - a Java only solution won't cut it (I've yet to come across a commercial Java system in the (MS-Windows based) businesses I've worked for over the years). Its got to be C# or (ugh) VB.NET (If there are GPL issues, then work with MONO at the very least).