JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 5, September-October 2003

The Impedance Imperative Tuples + Objects + Infosets =Too Much Stuff!

Dave Thomas,
Bedarra Corp., Carleton University and University of Queensland

THINGS ARE SO COMPLEX YOU NEED AN M.SC. TO PROGRAM CRUD!

Once upon a time it was possible for every new programmer to quickly learn how to write readable programs to **C**reate, **R**ead, **U**pdate and **D**elete business information. These so-called ***CRUD*** applications, along with reporting, were pervasive throughout business and essentially defined IT or MIS as it was called in those days.

These IT design patterns are so pervasive that they were incorporated into numerous useful *application program generators* for COBOL, PLI and RPG (Synon for example) and dominated the RAD market for well over a decade. Fourth generation languages such as Adabas Natural, PowerHouse, and Mapper were designed for CRUD applications operating over complex file structures and databases. Using these tools it was straightforward for a businessperson with minimal training to develop useful robust applications.

ALL YOU NEED IS TUPLES

The plethora of proprietary languages and file structures was a major enabler for the late Ted Codd's [1] relational model. SQL promised a single uniform abstraction called a "relational table" and three simple and powerful operations for all applications: "Select, Project and Join". The *expressive power and uniformity* of the relational model and SQL appeared very attractive to vendors who could then compete on engineering, rather than on the expressive power of their language and database structure.

SQL is quite good for simple CRUD applications on normalized tables. Unfortunately, SQL isn't computationally complete and often needs to be embedded within another programming language in order to build a CRUD application. These

Cite this column as follows: Dave Thomas: "The Impedance Imperative – Tuples + Objects + Infosets = Too Much Stuff!", in *Journal of Object Technology*, vol. 2, no. 5, September-October 2003, pp. 7-12. http://www.jot.fm/issues/issue_2003_09/column1

resulted in several embedded SQL dialects for different vendors that extended SQL beyond the standard. In order to be able to navigate through relations embedded SQL programming often requires an alternative interface using cursors (relation relative pointers) and materialized tuples in memory as record structures. -- Fortunately this hybrid embedded SQL was only used on the server but it requires special skills and often makes applications expensive to port from one vendor to another.

Despite the popularity of the relational model, it was often used as an implementation technology largely due to the need to have normalized tables. Information architects and business modellers used the more expressive and simpler Entity Relationship (ER) model (born again in UML). In order to implement a relational schema and associated operations, analysts and designers who work at the application level are forced to mechanically translate to an underlying relational model. It is surprising, given the utility of the Entity Relationship (ER) model, that there were few ER databases or at least ER languages other than ZIM.

For over a decade, relational technology remained unable to compete with the strong specific solutions of previous generations. This is not an uncommon lesson and one that often gets ignored by new technology zealots. Consider how long it will be before performance of a J2EE server meets or exceeds a previous generation TP monitor.

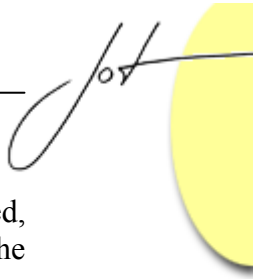
However, after many years of engineering, the relational databases can finally claim the performance and flexibility of keyed files (blobs); network databases (special support for transitive closures such as recursive bill of materials or CAD); and at least rudimentary support for text (XML in a Blob). Unfortunately each of these database and SQL extensions is quite proprietary, much in the way the early 4GLs were.

MINIMAL AFFORDANCES FOR OBJECTS AND CONTENT

In order to accommodate the increasing demand for objects and content, the DBMS vendors replied with the Third Generation Database Manifesto [3]. In particular they added new native types to the database to support objects (called user defined data types) and large text types. Both of these extended types were syntactic extensions on Blobs, which were added largely to support images, and documents. SQL was extended to allow query operations over Blobs using special content selector objects. Recently text types have been enhanced to support XML schemas or DTDs.

ALL YOU NEED IS OBJECTS

In the mid 80s, programming languages moved away from procedural languages to object languages. At this point the technical seam between objects and tuples became clearly exposed since objects always encapsulated their data and hence tuples or tables needed to be explicitly materialized as objects or tuples (objectified or tuplified). Object zealots pontificated the Object-Oriented Database Manifesto [2] touting the *expressive power*



and uniformity of objects (starting to sound familiar?). Object databases, it was claimed, solved the *impedance mismatch* between the object programming language and the database since everything was an object.

In the hopes of becoming the next Oracle, VC funded OODBMS vendors rushed to bring out object only databases such as Gemstone, Object Design, and Objectivity etc. Unfortunately for OODBMS advocates, while there are some solutions (AS/400 and Gemstone persistent stores) that have been very successful for application development, for the most part relational databases do CRUD and offer a simplicity and performance that has been impossible to match except in niche applications such as CAD.

WRAP IT AND MAP IT

In order to deal with the entrenched nature of relational data, the OO language community opted for wrapping the relational database as an object. This still left a world far too complex for most developers who were required to objectify or tuplify data as it moved between "object land" and "tuple land". Suffice to say that it is a non-trivial exercise to write an object program, which accesses a relational database. To address this issue, a number of researchers and vendors developed relational mapping frameworks such as TopLink, ADO etc. These mapping frameworks with their associated tools and wizards reduced the need for developers to understand the mapping details. However, to use them properly developers required an intimate knowledge of the framework.

The constant need to materialize and dematerialize objects is extremely inefficient. -- A lot of machine cycles are wasted using these wrappers and mappers. Indeed, we are fortunate that disks are so slow and CPUs are so fast. However, memory resident relational data is becoming commonplace with 64 bit machines and I doubt the overhead will be tolerable in the long term. In principle it should be possible to optimize through these frameworks, however this requires that the compiler have intimate knowledge of the mapper, the wrapper and potentially the database itself.

ALL YOU NEED IS <INFOSETS>

XML is rapidly moving beyond DTDs and documents to the brave new world of XML schemas and Infosets [9]. All of the major vendors have announced support for Xquery [10] product offerings of one form or another. XML Schema adds another declarative form to the impedance stack while Xquery introduces another sophisticated language for processing semi-structured information. Both the XML Infoset and Object proponents argue convincingly that the world is a tree, or potentially a digraph e.g. a purchase order contains the supplier and purchaser information followed by a variable length list of line items being purchased. Those who have just come up to speed on DTDs and XSLT will need to keep running to keep up with the XML train.

XQUERY – YET ANOTHER QUERY LANGUAGE

In order to provide a sound foundation for XML query and update processing of semi-structured information, the XQuery activity has defined a semantic model for XML based on Infoset [9]. It is too early to tell how well XQuery will be accepted and used, although major vendors are all showing products in plan which implement XQuery in one form.

This means that a developer will need to know one language for walking the tuples e.g., SQL, another for navigating the objects e.g., Java or C#, and a third for the Infoset e.g., XQuery. XQuery also assumes the knowledge of XPath[1] that is being refined in parallel to XQuery. While XQuery should provide the ability to directly query both tuples and infosets since in principle infosets are a superset of tuples it isn't clear that this can be done as elegantly as with SQL. It can be argued that given the ability to directly query both relational and XML data one can handle lots of problems without needing objects. The frustration with XQuery is the fact it is not a complete language, which means that once again developers must use multiple languages to achieve a simple CRUD application with a decent UI.

Unfortunately the lack of explicit XML values mean that parsing and generation tools always need to be present to interface between infosets and current OO programming languages. Nowhere is this more evident in SOAP and XML-RPC, the XML equivalents of procedural RPC and OO RMI. The complexity of serialization and unserialization of objects, and tuples to and from XML itself requires a special purpose toolkit.

While there are many critics who claim that XML processing can never be efficient, the same was said of relational databases and object technology. There are well known techniques for efficient serialization. Further recent research has demonstrated that most XML processing can be done without an inefficient memory based DOM model. Both events driven parsing and streaming are becoming the rage, although it has existed for years in content engineering languages [11] such as Omnimark [5], as alternatives to inefficient and the error prone debugging of XSLT.

[1] There is a still lot of discussion about infosets and schemas with some strong opponents of the current incumbents.

A TOOL OPPORTUNITY

It is clearly unnecessary to have three different ways to declare, navigate, communicate and convert values from these three solitudes. It is even worse if one chooses to use multiple OO languages, different DBMS vendors, and different XML schema processors. But this is not an unrealistic scenario for development of an enterprise application. Clearly the impedance problem gives rise to a tool opportunity.

Meta programming [11] or generative programming is clearly the least offensive way to cope with this mess. A model driven generator can clearly address the syntactic redundancy and associated mappings. The generator handles the syntactic redundancy. This however is the easy part. Processing is still far too complicated. Unfortunately, most generative tools do not support debugging at the level of the abstraction, forcing programmers to have deep knowledge of the generated code and the underlying framework.

SURELY WE CAN DO BETTER!

The impedance of incompatible type systems imposes a constant runtime overhead in addition to the syntactic burden. We need to move beyond the three solitudes and go beyond the gratuitous complexity that exists today. There is no reason we can't develop languages that are as productive and easy to use as 4GLs and which have underlying execution semantics based on integrated type system where tuples, objects and infosets are all first class. Microsoft .NET has already demonstrated that it is possible to have an OO language that can contain tuples and records as native types.

There are some early exemplars that show lots of promise. Xduce [6] provides an interesting semantics for Infosets and related research. Xtatic investigates XML as a native type in an OO language like C#. The recent research paper on the unification of tables, objects and documents [7] provides an interesting example of how an existing OO language such as C# or Java can be semantically and syntactically enhanced to address the problem.

We need to step back and consider the accidental complexity that arises when all of the various technical components are presented to a business developer. In isolation each of these technologies has clear merits but even a simple business application is far too complex when we compose the parts. We need to apply our considerable efforts to developing languages/tools as simple and useful for business users as 4GLs have been and continue to be. We need a computationally complete end user programming language, which will allows a mere mortal to create and deploy applications across a federated collection of semi-structured information.

REFERENCES

[1] F. Codd, "A Relational Model of Data for Large Shared Data Banks", *Communications of the ACM*, Vol. 13, No. 6, June 1970, pp. 377-387.

[2] Malcolm Atkinson, François Bancilhon, David DeWitt, Klaus Dittrich, David Maier, Stanley Zdonik, "The Object-Oriented Database System Manifesto", *Proceedings of the First International Conference on Deductive and Object-Oriented Databases* (1989)

[3] C. J. Date, Hugh Darwen, *Foundation for Future Database Systems: The Third Manifesto*, Second Edition, Addison-Wesley, 2000 (ISBN: 0-201-70928-7).

[4] Eric van der Vlist, *Comparing XML Schema Languages*, http://www.xml.com/pub/a/2001/12/12/schemacompare.html

[5] Mark Baker, *Internet Programming with OmniMark*, Kluwer Academic Publishers, Boston, ISBN 0-7923-7237-9, October 2000, 412 pp.

[6] Haruo Hosoya and Benjamin C. Pierce, Xduce, http://xduce.sourceforge.net/papers.html

[7] Erik Meijer and Wolfham Schulte, *Unifying Tables, Objects, and Documents*, http://research.microsoft.com/~emeijer/Papers/XS.pdf

[8] Dave Thomas, *Content Engineering – Time To Get Serious About Semi-Structured Information*, Otland September 2003.

[9] The XML Infoset, http://www.w3.org/TR/xml-infoset/

[10] D. Chamberlin, XQuery: "An XML query language", tutorial overview, *IBM Systems Journal* 41(4), 2002.

[11] Dave Thomas: "Reflective Software Engineering - From MOPS to AOSD", in *Journal of Object Technology*, vol. 1, no. 4, September-October 2002, pp. 17-26. http://www.jot.fm/issues/issue_2002_09/column1

About the author

Dave Thomas is CEO of Bedarra Corp., Adjunct Professor at Carleton University, Canada and University of Queensland, Australia, founding Director of AgileAlliance.com, and founder of Object Technology International. Bedarra works with research labs and commercial partners to transition innovations into products and practices.