ODBMS.ORG User Report No. 32/09
Editor Roberto V. Zicari- ODBMS.ORG *www.odbms.org*

September 2009.

Category: **Industry**
Domain: **Finance**
User Name: **Dr. Andreas Geppert**
Title **Platform Architect**
Organization: **Credit Suisse, Switzerland.**

I have been a platform architect at Credit Suisse since 2001. As a student (in the 1980s), I have joined one of the first OODBMS implementation projects (the DAMOKLES system). Since 1990, I have been a research assistant in the Database Technology Research Group (headed by Prof. Klaus R. Dittrich) at the University of Zurich. After having received my PhD in Computer Science from the University in Zurich in 1994, I have been a senior research assistant. From 1998 to 1999, I have been a visiting scientist at the IBM Almaden Research Center in San Jose, California. My research interests have been active and object-oriented information systems and information system integration. I continue to teach courses at the University of Zurich (a class on data warehousing and a hands-on database lab).

Q1. Please explain briefly what are your application domains and your role in the enterprise.
Credit Suisse is one of the world's largest banks and integrates Private Banking, Investment Banking, and Asset Management. Our application domains thus cover banking applications (in one of the three aforementioned areas), support applications (e.g., HR), and applications required to manage IT itself.
Many of the banking applications are home-grown, while in the other areas we try to purchase products. Our strategy also is to buy infrastructure where possible and not develop it in-house.

The role of Platform Architecture is to define so-called application platforms (pre-integrated technical components + processes + concepts, standards, and guidelines). Our strategy is to keep our technology portfolio as lean as possible, and one of the tasks of Platform Architecture is to position technologies for their use in the bank and devise concepts, standards, and guidelines for the technologies in use. In order to keep our portfolio as small as possible, we standardize on general-purpose DBMS and try to use them for different application areas (e.g., we use our Oracle build for transaction processing and data warehousing, where certain application areas may however need different additional options). In other words, we use special-purpose database systems only in restricted cases.
Within Platform Architecture, I am responsible for data warehousing and database technology.

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?

We certainly have an impedance mismatch problem, in particular as we are increasingly developing new applications in Java accessing relational databases such as Oracle and DB2.

My assumption however is that in our case the impendance mismatch is less dramatic as in other companies, i.e., the gap between data and object models is typically less big than in other application domains.
In particular, while we might have inheritance and specialization/inheritance hierarchies, we do not have highly complex structures as they can be met in engineering or manufacturing applications.

Q3. What solution(s) do you use for storing and managing persistence objects?
Relational (strategic are DB2 and Oracle) and still some hierarchical database management systems are used for storing data (in addition to files of course). Special-purpose DBMSs (e.g., multi-dimensional ones) are used in selected areas such as OLAP. Applications are implemented in Java or PL/1.

Q3 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)?
We have a bulk of experience with hand-coded JDBC. Obviously, there the mapping of the object model onto the data model is done by the developer (i.e., he/she resolves the impedance mismatch). In this approach, one has a high degree of flexibility, in particular in order to implement an efficient and performant mapping. On the negative side, there is no support for how to map objects onto relational data and the API is low-level (and hence error prone). It is also difficult to find good developers who are proficient in both, object-oriented programming and relational database technology.
One problem with JDBC is that developers have to code similar things over and over again, and that they therefore tend to develop their own frameworks on top of JDBC. This is something you would rather want to prevent from an architecture point of view; instead you would want to standardize on a single, general framework.

We have also considered to use SQLJ, which in comparison to JDBC is a bit more abstract, less error prone (e.g., more type safe) and potentially more performant (due to compile-time optimization). However, SQLJ skills are even harder to find than those for JDBC, and the adoption of SQLJ in the Java community is rather low.

We have furthermore considered the use of EJB 2.1 Entity Beans, but decided against it.

Recently, the Hibernate Object/Relational framework has been used in a few projects. Experience generally has been positive, in particular because there the mapping between the object model and the data model can be specified and the required JDBC code is generated by the framework. On the downside, one still has to monitor, investigate and eventually tune the generated SQL code in order to achieve good performance. We refrained from declaring Hibernate a company standard, because it is still proprietary and we would not take the risk of depending on a single provider.

Currently we are in the process of standardizing on the Java Persistence API (JPA), which is a part of EJB 3. Pros and cons are the same as for Hibernate, with the exception that JPA is an industry standard and several providers exist. Similar as for

Hibernate, SQL and JDBC are all but transparent for developers, and so much of what has been gained by specifying the mapping may have to be spent for tuning.

I do not think that there is significant experience with OODBMS-or ORDBMS-technology in the company.
Personally, I have practical experience with several OODBMS- (including O2, Poet, ObjectStore, Versant, Ontos) and ORDBMS-products (DB2, Oracle) and prototypes (Exodus, Ode, Damokles, etc.). O2 was a system that resolved the impedance mismatch almost perfectly. It has been the most beautiful database system I have ever worked with; particular highlights were the query language (OQL) and its integration into the programming language (O2C). The other systems lacked the complete support for OQL and/or its seamless integration into the programming language.
In object-relational DBMSs such as Oracle and DB2 (and the underlying standard, SQL:1999), I liked their support for type and table specialization and inheritance hierarchies and object references. IMHO their support for methods is less successful. In general, they do not really decrease the impedance mismatch, because applications still have to use JDBC or any other SQL-based API to access an (object-relational) database.

Q4. Do you believe that Object Database systems are a suitable solution to the "object persistence" problem? If yes why? If not, why?
Yes and no.
Yes, because O2 and other systems have shown that object-orientation and databases can be integrated almost seamlessly. In some application areas (such as engineering) in my opinion a full-fledged OODBMS is probably the best solution. Other systems such as ObjectStore did not resolve the impedance mismatch very well, and thus I think it still depends on the implementation.
There is currently a trend away from general-purpose DBMSs, and special purpose DBMSs are built for various application areas (e.g., data warehouse appliances, triple stores in the semantic Web). Maybe this means for OODBMS that they can find their accepted niches as well.

No, because OODBMSs are (in my opinion) not a general-purpose solution for all sorts of applications requiring persistence (including banking applications, or transaction processing in general).
What makes it further difficult for OODBMS is that they are conceived as a commercial failure in the IT community. We at Credit Suisse, for instance, follow the mainstream and apparent trends in the database area. In this mindset, people do not use OODBMS because nobody else uses them ...

Q5. What would you wish as new research/development in the area of Object Persistence in the next 12-24 months?
From my point of view, the most promising areas are EJB3 and the Java Persistence API (and related technologies for other environments). Even though underlying databases are still relational and thus the impedance mismatch is still there, it does (ideally) no longer concern the programmer, because the persistence provider (object/relational mapper) bridges it.
I would like to see more research and development in this area.
There should be additional persistence providers (JPA implementations) to foster competition and to base the JPA-standard on a broader base of vendors (currently, the main "vendors" are Eclipse, JBoss, and Apache).

Generation of efficient SQL in this context is another challenging topic.
The next topic is probably not strongly related to object persistence, but to databases in general. The question is how database management systems and their architectures are impacted by the recent development in the areas of solid state disk and flash memory.