Implementing the Executable Conceptual Model (ECM)

Dr. Reinhold Thurner

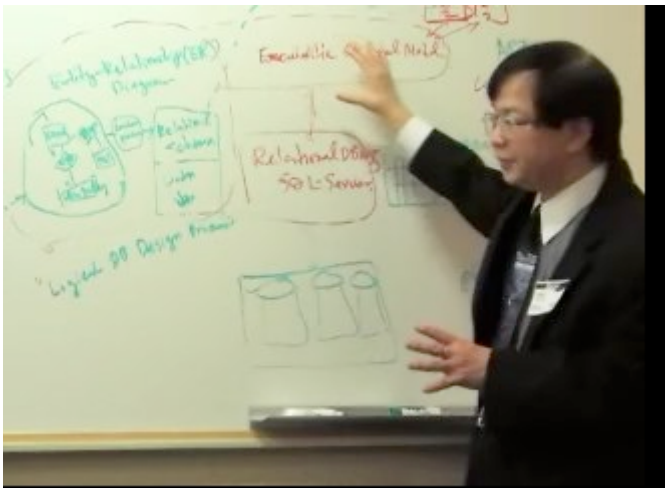
Rebweg 21, CH8700 Küsnacht, Switzerland

Q: What is an "executable conceptual model"?

The term was created by Peter Chen to describe a data infrastructure (database) which is directly based on the conceptual (i.e. Entity-Relationship) model. Conceptual models are the generally accepted approach to design abstract models of the real world. ECM takes this concept one step further and uses a conceptual model also at the instance level. The conversion of a conceptual model to tables of the relational model results in loss of semantics and creates the so called "impedance mismatch". In contrast to object-relational mappers the ECM does not try to bridge the impedance mismatch – it simply does not create it.

In April 2007 Peter Chen and Jose Blakely discussed this subject in a widely published video conference*). Peter Chen explained that the entity relationship model is not only a tool to design conceptual models but also a basis for the "execution of conceptual models". Jose explained the role of the conceptual models in the world of the relational model.

*)http://channel9.msdn.com/Shows/Going+Deep/Dr-Peter-Chen-Entity-Relationship-Model-Past-Present-and-Future



Q: What is the problem with conceptual modeling alone?

The designer's cookbook says: "The development of a conceptual model (with UML) is the first step in data modeling. This model is then transformed into a logical and physical model of a relational database. Then the database is set up and data can be loaded into the database." The modeling expert Michael Blaha recently explained "In practice, I usually construct a conceptual model with a UML tool. Then I rekey the model into a database tool". In practice the programmers don't "see" the conceptual model any more – they work directly with the physical model and subsequent changes in the structure of the data base are not reflected in the conceptual model. As a consequence the connection to the original conceptual model is lost and it becomes quickly outdated.

Q: This extends the ER-concept beyond modeling

Yes, we have to change the way how we are developing models. The current approach of a one-way-street from conceptual model to implementation does not work very well. It based on the wrong assumption that we can design a correct and final model without the implementation of data. With an "executable conceptual model" you design the model and test it with data. The data will tell you sooner or later that the model does not fit and you must change the model. With this integration the model and the data are always in sync.

Modeling is like drawing a sketch from a vaguely know landscape – then waking with the map in the landscape: if the map says "there is bridge" and reality tells “no bridge” it is advisable to believe reality and change the map. This approach represents is a more flexible modeling cycle and it goes on and on as long as the data and the application are in use. Ergo - an important and inseparable part of modeling is the instantiation (execution) of the model with real life data. We don't assume any more that the first cut of the conceptual model is perfect; we work with a first cut which is reasonably useful and leads to the next improved version.

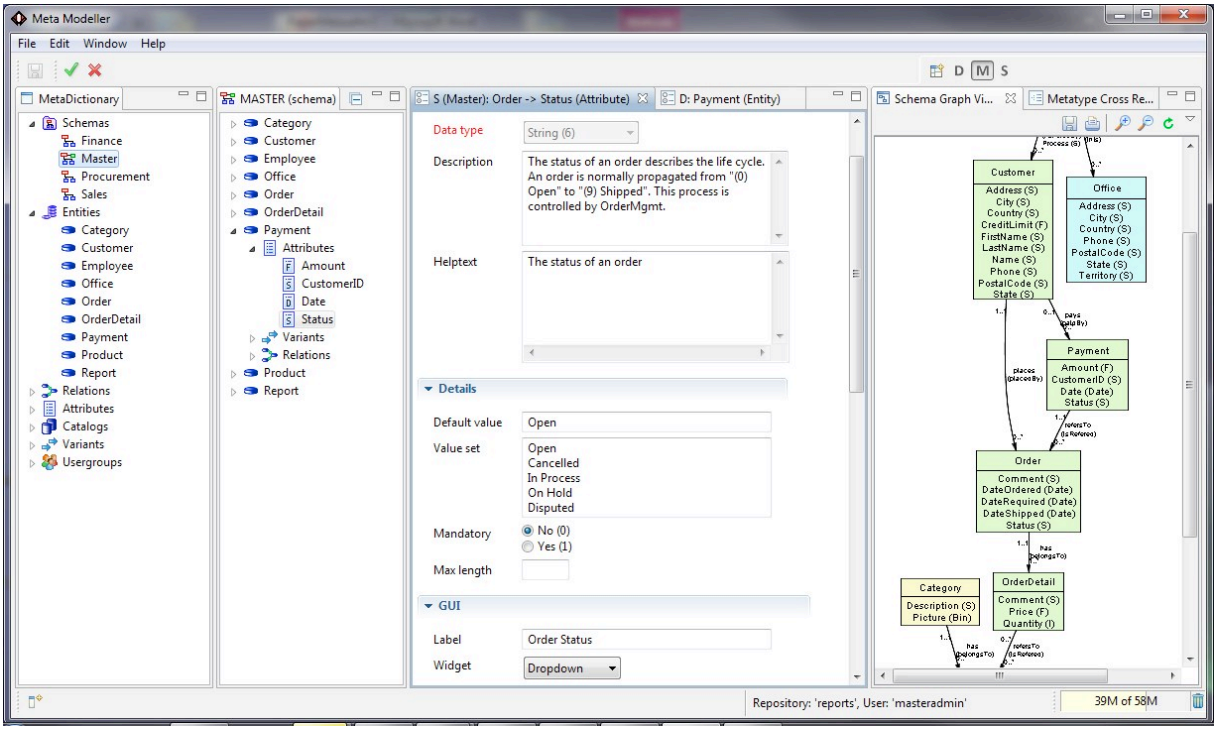
Q: Could you give a practical example

I may use the example Peter Chen and Jose Blakely used in their presentation. Peter talked of the conceptual view and said «In the real world we think of Persons who work in an Office-Building. This can be directly represented by entities (Person, OfficeBuilding) and the relationship “works in”. » The relational technology in contrast talks about a “Person-Table” and an “Address-Table” which contains the ID of the Person.

Using Metasafe you would define first the entity-types “Person” and “OfficeBuilding” and the relationship-type “worksIn” along with their properties in the Metasafe dictionary.

Then - to create the conceptual model- you drag the entity-types from the dictionary into the model pane and connect them using the relationships.

But you don't stop with the creation of the model. To check or use the model you can instantiate the entities “Miller” and “Lab21” and connect them via the relationship “worksIn”. “Relational integrity” is enforced because the core system would hinder you to delete an entity which is tied to another one via a relationship. If Miller leaves the company you would first delete the relationship "worksIn" (fairly obvious) before removing Miller as an entity.



Q: How is that supported by Metasafe

Metasafe is a DBMS with an Entity-Relationship (meta) model which integrates the conceptual model and the instance data. Metasafe is structured into four layers. The top layer (m3) – the metamodel is encoded in a 2000 line XML-boot-file. It describes the next layers and controls the modeling tools in a model-driven manner.

The m2-level consists of the dictionary and of the master model. The dictionary (you could also call it a glossary) contains a detailed description the entity-types, relationship-types, attribute-types, catalogs, variants etc. and their properties. These elements of the dictionary are used to construct the master model where entity-types are connected by bidirectional relationship-types and attribute-types are assigned to them. A modeler tool supports this modeling task and creates the corresponding diagrams. It is worth to mention that the modeler is controlled by the m3-model and runs under transaction control of the DBMS. The modeler uses the standard API, which can be used to build specific tools or create or extend the master model on the fly.

The m1-layer contains user or application specific models as subsets of the master model. These submodels are used to make very large models reasonably manageable and to grant specific access rights for users. These submodels are also conceptual models and are stored in the database.

The most important difference between classical modeling tools is the integration of the m0-level in Metasafe. The m0-level contains the actual instance data – i.e. instances of entities with their attributes and their relationships. The m0-level structures the data in catalogs to provide name spaces and manages versions and variants (or editions) of entities.

The system fulfills the ACID-criteria and provides full transaction support, fine grained access rights and combines the core (runtime system) with the necessary tools to manage and to access the database.

Q: What is the role of versions and variant in this context?

"For some applications, there is a need not only to store trees (data), but also to store the history of trees (data) as they evolve over time" (Patterns of Data Modeling by Michael Blaha, CRC

Press, 2010). This is the case e.g. in application development where the life cycle of artifacts must be monitored. The stages in the life cycle are identified by "variants" and access rights to individual variants of an artifact can be granted to user groups. Variants and revisions are orthogonal to the "entity-name". Basic access methods provide an easy access also to groups of an instance (e.g. all revisions of a given variant of an entity). This is handled directly by the core and not at application level.

Q: Does it also work on a larger scale?

There are still some limitations concerning the volumes of data. The present system can handle thousands of types and millions of instances, but not hundred millions; it can handle gigabytes of data, but not terabytes. In its present state a small implementation (using Windows/7 and derby) handles dozens of concurrent users. For larger populations and larger data volumes we would move to a proper server infrastructure. The system is portable across a large range of platforms and can be tailored to the needs of a practical application. However it is no match for the relational technology when it comes to very large volumes of data or concurrent users.

Q: Could it replace the Relational Technology?

Yes and No. It is No for the large business applications with high demands for performance and data volume. This may change with radically new concepts of physical data storage or diskless storage of data.

It is Yes if you require high flexibility, high complexity but lower volumes. There are thousands of applications in the low end of volume (n GB) and high end of complexity. This concerns frameworks (like COBIT, ITIL-CMDB, SPICE), general metadata repositories, it concerns tools for modeling or documentation of models of large software systems (SAP with nearly 10.000 Tables) and it concerns also small business applications. Well-known examples are “business applications” like reporting and auditing systems. They are complex they require a high degree of flexibility and suffer from frequent changes. Today the majority of these applications are “written in Excel”. An “executable conceptual model” is by far the better solution for this kind of applications.

Q: How does ECM compare with the "MS Entity Framework"

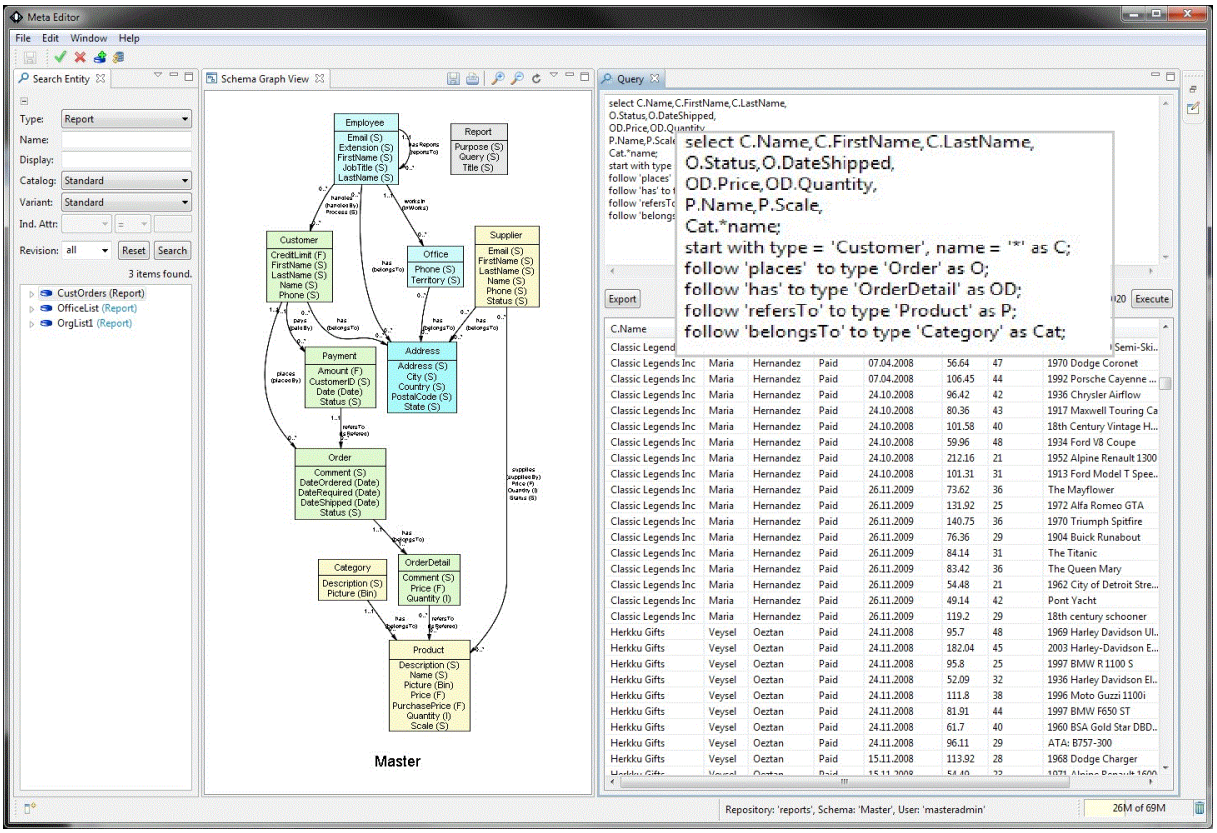
We have the same goal – to “eliminate the impedance mismatch for both application and data services" as Jose Blakely described it in his paper “next Generation Data Access, Making the Conceptual Level Real”. The Entity Framework however must live with the additional restriction to stay compatible with an existing "eco-system of relational data bases".

The Entity Framework– similar to hibernate – is an object-relational mapper. Tools are provided to create the mapping between the conceptual model and the implementation on a relational DBMS. The tools of the EF generate the corresponding code and (relational) data base definitions from this specification. Consequently the Entity Framework has all the advantages of compatibility with an RDBMS but has to pay a price in terms of structural limitations and intermediate generation steps. The direct implementation of the conceptual model (ECM) does not crated this impedance mismatch does not require any manual mapping and intermediate code generation. It has to pay the price of incompatibility with an RDBMS (not so much with the real world). The practical difference becomes apparent when you start to extend the model: in Metasafe you can extend the model (using the modeler), commit the changes and in the next minute enter data into the database.

Q: Is there an access language comparable to SQL or LINC

The system core has an API which interprets an "erSQL" query and delivers the required objects: Similar to LINC it returns a list of typed objects and not only a flat table of strings. The erSQL syntax is is self-explanatory and directly related to the conceptual model. A program can invoke such a query and process the information delivered by the query. A good example for such a program is the data driver for BIRT (Business Intelligence and Reporting Tool) which provides an easy means to create reports and BI-analyses.

The example below shows how a query is derived directly from the conceptual model.



Q: You called Metasafe also a "Repository"

This name exists for historical reasons. When we started with a first version (in C) we were still thinking of the system as a repository infrastructure – similar maybe to ASGs Rochade. This is also the reason why we implemented Catalogs, Versions and Variants of entities right from the beginning. A powerful and state-of-the-art core is certainly important. But the accompanying tools are at least as important for such a system. The development of these tools in C was a real nightmare. This changed radically when we moved to a complete reimplementation with Java, Eclipse and model-driven development of tools. With this technology the system has outgrown the former limitations and a wide range of applications can be supported beyond metadata repositories.

Q: How important are tools really

A basic set of tools is important for two reasons: First it helps the user to get immediately started with his job. Second it helps us to understand what kind of services the core should provide. This contributes a great deal to the improvements of the core API.

Metasafe provides several general purpose tools: The metaModeler is an Eclipse based tool for the development, maintenance and documentation of models. The metaEditor is an Eclipse package and provides various views (Plugins) to browse, display, edit, import and export data or to create

graphical representations. The tools are controlled by the model in a model-driven fashion. The behavior of the tools is completely controlled by the structure of the user model and the granted access rights. Data can be imported from and exported to external sources like Excel, XML, flat files or SQL-databases. The tools are all developed as Eclipse plugins and can be extended or adjusted to individual requirements.

Q: Why did you choose Java for the implementation?

For some time we kept two versions – Java and C# - in parallel. But then we had to decide which framework we would use and we decided to choose Eclipse and Java. To create a version hooked into DotNet would not pose insurmountable problems.

Q: What are your next steps?

We have developed the system in a small group with several cycles of refinement, optimization and practical test cases and proof of concepts. The system has now reached a level of functionality and quality where it can be used on a broader scale. Therefor we are looking for interesting parties to take it further into different application areas. A repository for application development and application life cycle management could be such an area. Metasafe can replace thousands of tiny text files (XML, script, doc etc.) to manage the information in a transparent and secure manner. We were also discussing the idea to take the concept of the "Common Warehouse Meta Model" one step further to the instance level. The system has reached the level of quality and perfection to move to the next step and to work with interested parties.

_____

Reinhold Thurner is founder of the Metasafe GmbH in Switzerland, which developed an entity-relationship based DBMS. His main interest is in software engineering, modeling and representation of information systems, sematic models and development tools.

Since 2002 he is active as member of the board and of a banking service center. He held various positions as system architect in the banking industry especially in projects to restructure or reconstruct core banking systems. For several years he served as the president of the Swiss Computer Society (then named SVD).

Before he founded a software company (Delta) which developed an application generator for portable applications. The system was used by about 500 mostly large institutions to develop large batch and online-TP applications mostly for mainframes. In 1989 he sold the company.

He received his doctorate from the University of Zürich with a thesis on the implementation of Operations-Research-Algorithms. Then he developed systems software for Unisys computers and economic models. He lectured on software engineering at the University of St.Gallen.