

Mapping-Driven Data Access

Position Paper

August 23, 2006

Sergey Melnik
Microsoft Research, USA
melnik@microsoft.com

1. INTRODUCTION

Virtually all modern enterprise applications manipulate data represented in multiple formats, such as objects in a programming language, rows in relational databases, or XML structures, which are exposed through distinct programming models. As a result, application developers face a constant challenge of translating data and data access operations across different data representations. Even in simple object-to-relational mapping scenarios where a set of objects is partitioned across several relational tables we can end up with transformations that require outer joins, nested queries, and case statements in order to reassemble objects from tables (we will see an example of that shortly). No wonder, implementing such transformations in real applications is difficult, especially if data needs to be updatable. Supporting updates is a common requirement, since many enterprise applications manage operational data.

A common way of shielding the business logic from the intricacies of data manipulation is by using a *data access layer* that encapsulates the necessary transformations. Typically, the job of a data access layer is to provide an *updatable client-side view* that exposes persistent data as business entities. The backbone of a general-purpose data access layer is a *mapping* that establishes a relationship between data represented according to different schemas and data models. In contrast to hardwiring the data access code for a specific application, a mapping is used to parameterize the transformation logic or generate it automatically. Today, mappings between different formats are deployed as first-class programming artifacts in enterprise systems. Therefore, *mapping-driven data access*, i.e., the problem of translating data and data access operations over mappings, is of critical importance to the developers of enterprise applications.

Solutions to this problem have proven to be elusive, measured both in terms of commercial successes and elapsed research effort. The industry landscape is covered with carcasses of failed data access products and in-house projects. Products that made their way to the market offer widely varying degrees of capability, robustness, and total cost of ownership. Industry experts give mixed advice about purchasing off-the-shelf solutions vs. developing home-grown ones.

What is so hard about mapping-driven data access? Consider object-to-relational mappers (O/R-Ms), a widely used data access technology. It is relatively straightforward to build an O/R-M that uses a one-to-one mapping to expose each row in a relational table as an object, especially if no declarative data manipulation is required. However, as more complex mappings, set-based operations, performance, multi-DBMS-vendor support, and other requirements weigh in, ad hoc solutions quickly grow out of hand. This is not surprising since mapping-driven data access lies at the intersection of two longstanding data management problems:

- *Impedance mismatch*, the problem of accessing persistent storage through programming language abstractions. Its focus is on bridging the gap between two distinct data models (e.g., relational and object-oriented), usually with minimal data reshaping.
- *Data integration*, the problem of providing unified access to heterogeneous data. Here, the focus is on data reshaping, usually within a common data model, assuming that the impedance mismatch has been resolved (e.g., using wrappers).

The reality of building enterprise applications is that impedance mismatch and data integration problems go hand in hand. Solutions that address language binding but ignore data reshaping fall short when it comes to accessing legacy databases, or when applications and data need to evolve independently. Solutions that target data reshaping and leave language binding out of scope push a lot of plumbing work back to the developers.

2. TECHNICAL CHALLENGES

Decade after decade, database technology has been falling behind in providing adequate support to the developers of data access solutions [1, 9, 10, 15]. Using off-the-shelf mechanisms offered by a modern DBMS such as views, triggers, and stored procedures is difficult for a number of reasons. First, commercial database systems offer very limited support for updatable views. For example, views containing joins are usually not updatable. Second, defining custom database views and triggers for every application that needs to access mission-critical enterprise data is often unacceptable due to security and manageability risks. Moreover, support of rich modeling constructs used in business applications, such as inheritance and complex types, varies significantly from one DBMS to the next.

As a result, developers of data access solutions face a number of technical challenges:

1. *Mapping specification*. There exists no practical, general-purpose mechanism for specifying query and update behavior in mapping-driven data access scenarios. In many commercial products the mapping is represented as a custom data structure that has vague semantics and drives case-by-case reasoning. The ones that use a view-based approach (e.g., [8]) face update ambiguity, or fall back to highly procedural, error prone solutions such as custom update routines, INSTEAD OF triggers, etc. Yet other products do not support updates [17].
2. *Mapping runtime* for query and update processing. Case-by-case reasoning yields a very complex runtime that is hard to develop and maintain. It is fragile since adding a mapping

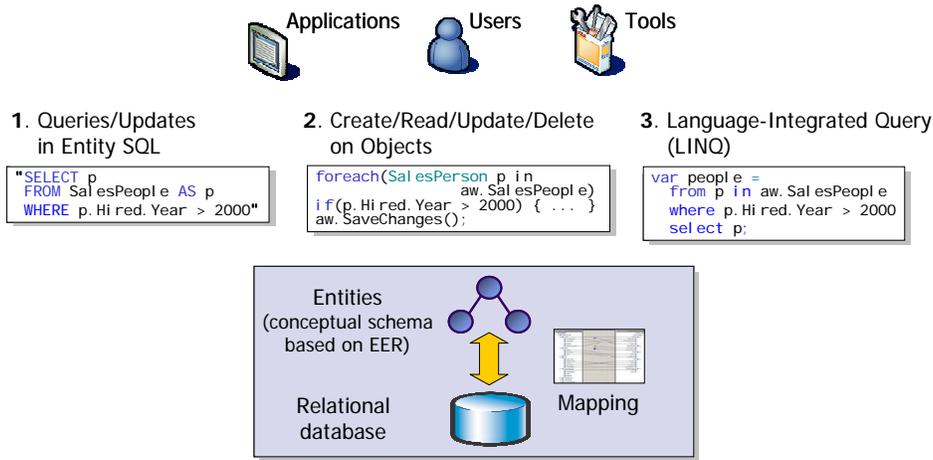


Figure 1: Mapping-driven data access in ADO.NET Entity Framework

feature may affect the entire code base, often with unpredictable ramifications. These difficulties multiply given that a runtime may need to support caching, prefetching, various concurrency and security models, etc. The challenge is to build a robust runtime that uses uniform algorithms.

3. *Mapping tools.* A data access layer is difficult to use without a tools suite that supports mapping creation, generation of programming interfaces, schema evolution, reverse-engineering of an existing database schema, tuning the persistent representation based on application workload, etc. Building such tools is very hard [3].
4. *Higher-level data services.* A data access layer needs to support a rich set of services such as reporting, replication, data mining and analysis, batch loading, and distributed transactions, some of which are easier and some are harder to build in the presence of a mapping.

Finding elegant solutions to the above challenges requires a major R&D investment across industry and academia.

3. UPDATES THROUGH MAPPINGS

The database research community contributed many powerful techniques that can be leveraged for supporting mapping-driven data access. And yet, there remain significant discrepancies between engineering practice and the academic research agenda. Among the most critical ones is supporting updates through mappings. Compared to queries, updates are far more difficult to deal with as they need to preserve data consistency across mappings, may trigger business rules, and so on.

Updates through database views have a long history. In 1978, Dayal and Bernstein [11] put forth the view update problem. They observed that finding a unique update translation over even quite simple views is rarely possible due to the intrinsic underspecification of the update behavior by a view. Subsequent research followed two major directions. One line of work focused on determining under what circumstances view updates can be translated unambiguously, most recently for XML views [7]. Unfortunately, there are few such cases; besides, in mapping-driven business applications every update needs to have a well-defined translation, i.e., rejecting a valid update at runtime is unacceptable.

Another line of research focused on closing the underspecification gap. A number of mechanisms were suggested in the literature,

such as constant complement [2] or dynamic views [12]. This work turned out to be mostly of theoretical value. Recently, researchers showed rekindled interest in the view update problem (see e.g. [6, 14, 16]). Still, to date there is no elegant and widely accepted mechanism for specifying view update behavior and processing updates in mapping-driven data access scenarios. As a consequence, commercial database systems offer poor support for updatable views.

In mapping-driven scenarios, the updatability requirement goes beyond a single view. For example, a business application that manipulates Customer and Order entities effectively performs operations against two views. Sometimes a consistent application state can only be achieved by updating several views simultaneously. Case-by-case translation of such updates may yield a combinatorial explosion of the update logic. Delegating its implementation to enterprise developers is extremely unsatisfactory because it requires them to manually tackle one of the most complicated parts of data access.

4. NOVEL MAPPING ARCHITECTURE

The database architects and researchers at Microsoft have been working on an innovative mapping architecture which aims to address the challenges identified in the previous sections. It exploits the following ideas:

- Mappings are specified using a *declarative language* that has well-defined semantics and puts a wide range of mapping scenarios within reach of non-expert users.
- Mappings are compiled into *bidirectional views*, called query and update views, that drive query and update processing in the runtime engine.
- Update translation is done using a general mechanism that leverages *view maintenance*, a mature database technology [5, 13].

The new mapping architecture enables building a powerful stack of mapping-driven technologies in a principled, future-proof way. Moreover, it opens up interesting research directions of immediate practical relevance.

The remainder of this paper gives a brief overview of the above mapping approach. It has been implemented¹ in the upcoming

¹A community technology preview of the ADO.NET Entity Framework is available for download at <http://msdn.microsoft.com/data>

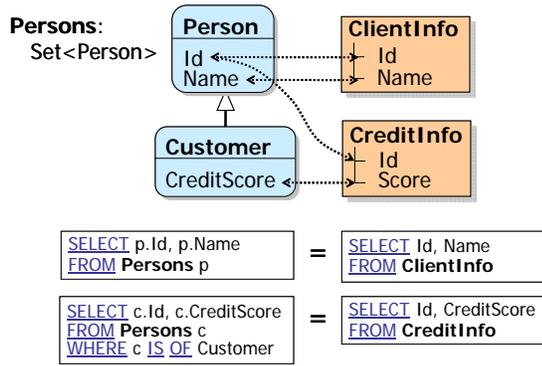


Figure 2: Mapping between an entity schema (left) and a database schema (right)

release of the ADO.NET Entity Framework, which provides a mapping-driven data access layer for .NET applications. The Entity Framework consists of an extended entity-relationship data model and a set of design-time and run-time services that allow developers to describe the application data and interact with it at a ‘conceptual’ level of abstraction that is appropriate for business applications [4]. The interaction modes include (1) declarative queries and updates expressed in Entity SQL (eSQL), an extension of SQL that can deal with inheritance, complex types, etc., (2) create/read/update/delete operations on objects, which expose a programming surface for business entities, and (3) LINQ, a language-integrated query mechanism that enables compile-time checking of queries (see Figure 1).

Mappings. A mapping is specified using a set of mapping fragments. Each mapping fragment is a constraint of the form $Q_{Entities} = Q_{Tables}$ where $Q_{Entities}$ is a query over the entity schema (on the client side) and Q_{Tables} is a query over the database schema (on the store side). A mapping fragment describes how a portion of entity data corresponds to a portion of relational data. That is, a mapping fragment is an elementary unit of specification that can be understood independently of other fragments.

To illustrate, consider the sample mapping scenario in Figure 2. It depicts a simple entity schema with entity types `Person` and `Customer` whose instances are accessed through a collection `Persons`, also referred to as an entity set. On the store side there are two tables, `ClientInfo` and `CreditInfo`. The mapping is given by two fragments shown in Figure 2. The first fragment specifies that the set of (`Id`, `Name`) values for all entities in `Persons` is identical to the set of (`Id`, `Name`) values retrieved from the `ClientInfo` table. Similarly, the second fragment tells us that (`Id`, `CreditScore`) values for all `Customer` entities are stored in the `CreditInfo` table. The mapping can be defined using an XML file or a graphical tool.

Bidirectional views. The mappings are compiled into bidirectional eSQL views that drive the runtime. The *query views* express entities in terms of tables, while the *update views* express tables in terms of entities. Update views may be somewhat counterintuitive because they specify persistent data in terms of virtual constructs, but as we show later, they can be leveraged for supporting updates in an elegant way. The generated views ‘respect’ the mapping in a well-defined sense and have the following properties:²

- $Entities = QueryViews(Tables)$

²The presentation is slightly simplified. In particular, the persistent state is not completely determined by the virtual state.

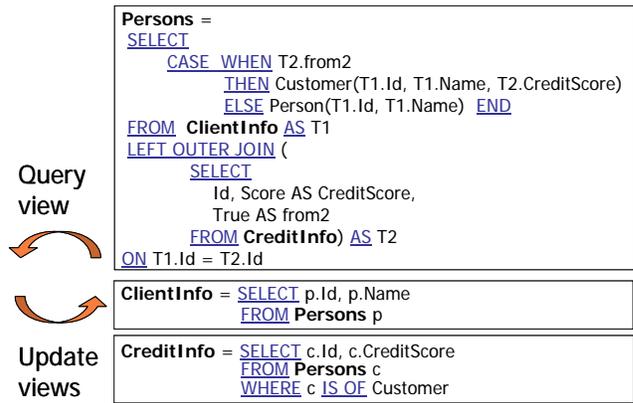


Figure 3: Bidirectional views compiled from mapping in Figure 2

- $Tables = UpdateViews(Entities)$
- $Entities = QueryViews(UpdateViews(Entities))$

The last condition is the so-called *roundtripping criterion*, which ensures that all entity data can be persisted and reassembled from the database in a lossless fashion. The mapping compiler included in the Entity Framework guarantees that the generated views satisfy the roundtripping criterion. It raises an error if no such views can be produced from the input mapping.

Figure 3 shows the query and update views generated by the mapping compiler for the mapping in Figure 2. In general, the views are significantly more complex than the input mapping, as they explicitly specify the required data transformations. For example, to reassemble the `Persons` entity set from the relational tables, one needs to perform a left outer join between `ClientInfo` and `CreditInfo` tables, and instantiate either `Customer` or `Person` entities depending on whether or not the respective tuples from `CreditInfo` participate in the join. Writing query and update views by hand that satisfy the roundtripping criterion is tricky and requires significant database expertise; therefore, currently the Entity Framework only accepts the views produced by the built-in mapping compiler.

Update translation. One of the key insights exploited in the ADO.NET mapping architecture is that view maintenance algorithms can be leveraged to propagate updates through bidirectional views. This process is illustrated in Figure 4. Tables hold persistent data. Entities represent a virtual state since typically only a tiny fraction of data is materialized on the client. The goal is to translate an update $\Delta Entities$ on the virtual state of `Entities` into an update $\Delta Tables$ on the persistent state of `Tables`. This can be done using the following two steps:

1. View maintenance:
 $\Delta Tables = \Delta UpdateViews(Entities, \Delta Entities)$
2. View unfolding:
 $\Delta Tables = \Delta UpdateViews(QueryViews(Tables), \Delta Entities)$

In Step 1, view maintenance algorithms are applied to update views. This produces a set of so-called delta expressions, $\Delta UpdateViews$, which tell us how to obtain $\Delta Tables$ from $\Delta Entities$ and a snapshot of `Entities`. Since the latter is not fully materialized on the client, in Step 2 view unfolding is used to combine the delta expressions with query views. Together, these steps give us an expression that takes as input the initial persistent data-

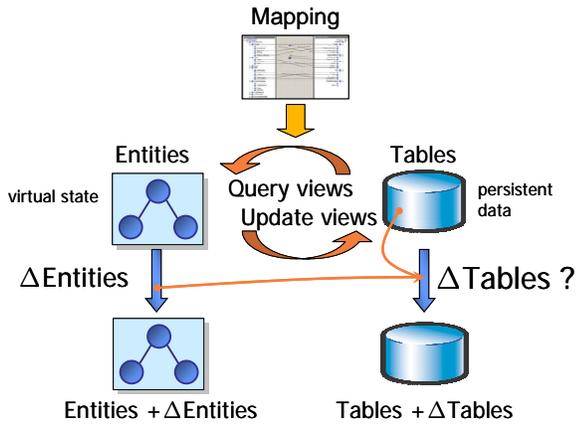


Figure 4: Bidirectional views and update translation

base state and the update to entities, and computes the update to the database.

This approach yields a clean, uniform algorithm that works for both object-at-a-time and set-based updates (i.e., those expressed using data manipulation statements), and leverages robust database technology. In practice, Step 1 is often sufficient for update translation since many updates do not directly depend on the current database state; in those situations we have $\Delta\text{Tables} = \Delta\text{UpdateViews}(\Delta\text{Entities})$. If $\Delta\text{Entities}$ is given as a set of object-at-a-time modifications on cached entities, then Step 1 can be further optimized by executing view maintenance algorithms directly on the modified entities rather than computing the $\Delta\text{UpdateViews}$ expression.

To illustrate, consider the update views in Figure 3. These views are very simple, so applying view maintenance rules is straightforward. We obtain the following $\Delta\text{UpdateViews}$ expressions:

```

ΔClientInfo = SELECT p.Id, p.Name FROM ΔPersons p
ΔCreditInfo = SELECT c.Id, c.Name FROM ΔPersons c
                WHERE c IS OF Customer

```

So, if $\Delta\text{Persons} = \{\text{insert}(\text{Customer}(1, \text{'Alice'}, 700))\}$, we get $\Delta\text{ClientInfo} = \{\text{insert}(\text{row}(1, \text{'Alice'}))\}$, $\Delta\text{CreditInfo} = \{\text{insert}(\text{row}(1, 700))\}$. In general, update views can get significantly more complex. Using view maintenance techniques allows supporting a very large class of updates and accounts in a uniform way for tricky update translations where multiple entity sets are updated, a deletion from an entity set becomes an update in the store, etc.

5. CONCLUSIONS

Mapping-driven data access is a fundamental problem in enterprise data management, which poses a number of hard technical challenges. The ADO.NET Entity Framework leverages innovative techniques that address many of those challenges. The mapping architecture deployed in the Entity Framework exposed interesting research issues concerning mapping compilation, checking roundtripping, enforcing data consistency using a combination of client-side and server-side constraints, exploiting efficient view maintenance techniques for object-at-a-time and set-based updates, grouping updates to optimize performance, and many others.

6. REFERENCES

- [1] M. P. Atkinson, P. Buneman. Types and Persistence in Database Programming Languages. *ACM Comput. Surv.*, 19(2):105–190, 1987.
- [2] F. Bancilhon, N. Spyratos. Update Semantics of Relational Views. *ACM Transactions on Database Systems*, 6(4):557–575, 1981.
- [3] P. A. Bernstein. Applying Model Management to Classical Meta-Data Problems. In *Conference on Innovative Data Systems Research (CIDR)*, 2003.
- [4] J. A. Blakeley, D. Campbell, J. Gray, S. Muralidhar, A. Nori. Next-Generation Data Access: Making the Conceptual Level Real. At <http://msdn.microsoft.com/data/default.aspx?pull=/library/en-us/dnvs05/html/nxtgenda.asp>, June 2006.
- [5] J. A. Blakeley, P.-Å. Larson, F. W. Tompa. Efficiently Updating Materialized Views. In *ACM SIGMOD International Conference on Management of Data*, 1986.
- [6] A. Bohannon, B. C. Pierce, J. A. Vaughan. Relational Lenses: A Language for Updatable Views. In *ACM Symposium on Principles of Database Systems (PODS)*, 2006.
- [7] V. P. Braganholo, S. B. Davidson, C. A. Heuser. From XML View Updates to Relational View Updates: Old Solutions to a New Problem. In *International Conference on Very Large Data Bases (VLDB)*, 2004.
- [8] M. Carey, the AquaLogic Data Services Platform Team. Data Delivery in a Service-Oriented World: The BEA AquaLogic Data Services Platform. In *ACM SIGMOD International Conference on Management of Data*, 2006.
- [9] M. J. Carey, D. J. DeWitt. Of Objects and Databases: A Decade of Turmoil. In *International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann, 1996.
- [10] W. R. Cook, A. H. Ibrahim. Integrating Programming Languages & Databases: What's the Problem? Draft at <http://www.cs.utexas.edu/~wcook/Drafts/2005/PLDBProblem.pdf>, Oct. 2005.
- [11] U. Dayal, P. A. Bernstein. On the Updatability of Relational Views. In *International Conference on Very Large Data Bases (VLDB)*, 1978.
- [12] G. Gottlob, P. Paolini, R. Zicari. Properties and Update Semantics of Consistent Views. *ACM Transactions on Database Systems*, 13(4):486–524, 1988.
- [13] A. Gupta, I. S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.
- [14] Y. Kotidis, D. Srivastava, Y. Velegrakis. Updates Through Views: A New Hope. In *Intl. Conf. on Data Engineering (ICDE)*, 2006.
- [15] T. Neward. The Vietnam of Computer Science, June 2006. Blog article at <http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx>.
- [16] B. C. Pierce. The Weird World of Bi-Directional Programming, Mar. 2006. ETAPS Invited Talk. Slides at <http://www.cis.upenn.edu/~bcpierce/papers/lenses-etapsslides.pdf>.
- [17] M. Roth, M. A. Hernandez, P. Coulthard, L. Yan, L. Popa, H. C.-T. Ho, C. C. Salter. XML Mapping Technology: Making Connections in an XML-Centric World. *IBM Systems J.*, 45(2), 2006.