Worry-Free Database Upgrades: Automated Model-Driven Evolution of Schemas and Complex Mappings

James F. Terwilliger
Microsoft Corporation
james.terwilliger@microsoft.com

Philip A. Bernstein
Microsoft Research
phil.bernstein@microsoft.com

Adi Unnithan
Microsoft Corporation
adi.unnithan@microsoft.com

ABSTRACT

Schema evolution is an unavoidable consequence of the application development lifecycle. The two primary schemas in an application, the client conceptual object model and the persistent database model, must co-evolve or risk quality, stability, and maintainability issues. We present MoDEF, an extension to Visual Studio that supports automatic evolution of object-relational mapping artifacts in the Microsoft Entity Framework. When starting with a valid mapping between client and store, MoDEF translates changes made to a client model into incremental changes to the store as an upgrade script, along with a new valid mapping to the new store. MoDEF mines the existing mapping for mapping patterns which MoDEF reuses for new client artifacts.

Categories and Subject Descriptors

D.2.2 [**Design Tools**]: Evolutionary Prototyping; D.2.12 [**Interoperability**]: Data Mapping

General Terms

Algorithms, Theory

Keywords

Schema evolution, model management, O-R mapping

1. INTRODUCTION

Object-Relational Mapping systems (ORMs) have become a popular, if not essential, tool for programmatic access to persistent data. In any ORM system, there are three artifacts: a model of the client (usually including object inheritance), a model of the store (i.e., a relational database), and a mapping between the two. As an application evolves over its life cycle, its client model may change, so the store model and mapping must adapt as well both to maintain the validity of the mapping and to ensure that sufficient constructs are available in the store to persist client objects.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

Some ORM tools have infrastructure that handles automatic versioning, a key part of the evolution problem. Ruby on Rails [6] has migration features that allow an application that expects version X but detects a database with version Y to automatically invoke a script that migrates the database from version Y to version X.

Our system solves an orthogonal problem: how to generate the versioning scripts. The mapping used by Active Record in Rails is trivial; the model and database are always version aligned because the model is inferred by convention from the database. If the ORM supports non-trivial mappings between the client and store (e.g., [2, 5]), it takes manual effort to determine a proper store migration for a given client model evolution. Non-trivial mappings arise when the mapping language is flexible, thereby allowing the DBA to optimize performance by choosing the best mapping pattern for a workload. The more complicated the pattern, the more delicate the task is of choosing a store migration that will preserve the pattern when new constructs are added.

In this demonstration, we present MoDEF (Model-Driven Entity Framework), a Microsoft Visual Studio extension. Visual Studio ships with a designer to construct client models for use with the Entity Framework (EF), an ORM with flexible mapping capabilities [5]. The designer allows a developer to build a client model and manually map it to an existing store, or to use a pre-defined mapping scheme to generate a new database schema. Our extension captures changes made to a client model and updates the mapping and store model without additional user input. The result is a new model that correctly persists the new client model, a new valid mapping derived from the previous mapping and consistent with its mapping patterns, and a script that upgrades a database instance in-place. Thus MoDEF enables true model-driven design, where the model evolves during application development.

A prominent feature of the Entity Framework is its flexible mapping language with firmly grounded semantics. Given that flexibility, each construct that is added to a client model can have many different valid mappings to storage. A distinguishing feature of MoDEF is that it can mine an O-R mapping for patterns, and then create storage and mappings for new client constructs that are consistent with those patterns. This feature enables a user to add new constructs to a client model without specifying the mapping, regardless of the complexity of the existing mapping to storage.

Because client-side changes are tracked as they occur, MoDEF can provide more informed schema evolutions than would be possible by taking a difference between client ver-

sions. For instance, MoDEF recognizes entity type and property renaming as it occurs in Visual Studio when initiated by the developer. A tool that compares two different versions of a client model would only recognize that one entity type or property had been removed from the previous version, while a new one was added to the new version. As a more complex example, MoDEF allows one to refactor a client model by moving a property across inheritance, which would ordinarily be also recognized as a dropped and added property and result in a script that drops data.

In a large company and for large applications, a database administrator may be the final gateway to the product. The upgrade script produced by MoDEF is punctuated by comments that correlate fragments of the script with the client model change that generated it, thus conveying intent to a DBA who may want to tune the script and enabling communication between the DBA and the developer. As the requirements for an application change, MoDEF allows the developer to modify the application as needed, then provide the DBA with the necessary scripts and provenance information to manage the database effectively as well.

In Section 2, we outline our demonstration of MoDEF with model-driven schema evolution scenarios. We give a brief overview of EF and technical details of MoDEF in Section 3. Section 4 describes an example client-driven evolution. Finally, Section 5 covers related and future work.

2. WHAT WILL BE DEMONSTRATED

The demo shows how MoDEF automatically handles several typical model and mapping evolution scenarios. It begins with one of three different starting points:

- Empty models and mapping, which arises when a new application is constructed entirely model-first
- A client model and mapping generated by reverse-engineering a database, such as when an application is constructed from an existing database [3]
- A hand-created mapping from an existing client to an existing store for optimal physical performance

Regardless of the starting point, we begin the demo with a client model, a store model, and a valid mapping between them. We then lead MoDEF through several typical model-driven evolution activities, including the following:

- Add a new entity type, either as a child of an existing type or the root of a new hierarchy
- Add a new association between entity types
- Add a new property to an entity type, both where the property is nullable and where it must have a value
- Rename an existing entity type or property
- Drop an existing entity type, property, or association
- Refactor a one-to-one association between two entity types into an inheritance relationship (Figure 1)
- Move an entity type property to its parent or to a child

In the case of adding a new non-nullable property to an entity type, we assign a default value that is set for all existing instances of entity types. This default need not be the same as the ordinary default value that is set for new instances. For clarity, we call the value assigned to new instances the *default value*, and the value assigned to existing instances the *inherited value*.

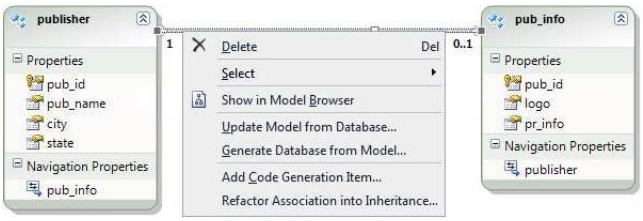

Figure 1: Refactoring association into inheritance

MoDEF keeps a log of actions as they occur on the Entity Framework design surface. The log is persisted to disk whenever the active model is saved; if Visual Studio is closed without saving changes, the log is discarded. When the developer is ready to construct a new version of the database, the developer selects the "Generate database from model" menu option, at which time the developer can choose to generate a new database or create an upgrade script from the log. MoDEF then processes the change log items, which are atomic actions expressed against the client model, and translates them into changes against the store model and the mapping. The next section describes these actions and their translation in more detail.

3. TECHNICAL DETAILS

EF is an ORM system whose mappings are equations between queries. Given a client (object) model C and a store (relational) model S, a mapping M is expressed as a collection of *mapping fragments*. Each mapping fragment takes the form "$Q_C = Q_S$," where Q_C and Q_S are select-project relational algebra queries over C and S respectively. EF compiles mapping fragments into views that describe how to move data between client and store schemas.

EF mappings are flexible enough to handle the three main hierarchy mapping schemes: Table-per-Type (TPT), Table-per-Concrete Class (TPC), and Table-per-Hierarchy (TPH). EF also allows multiple schemes to be used within a single hierarchy, hybrid approaches that blend the paradigms, and horizontal and vertical partitioning of client entity sets.

When adding a new property, entity type, or association to a client model, there are many possible mappings to persist the new client data. For example, suppose one adds entity type E to a hierarchy currently mapped entirely using a TPT scheme. One can map E to its own table without re-mapping inherited properties (consistent with TPT), or re-map properties inherited from only E's parent (a TPT-TPC hybrid), or map E to the same table as E's parent (thus creating a small TPH scheme of just E and its parent).

MoDEF automatically maps new client model constructs in a fashion consistent with existing mapped constructs, which the user can override if desired. If there is no single consistent mapping scheme among mapped constructs, MoDEF examines schema elements that are the most "similar" to the new construct. For a newly added entity type E, MoDEF assigns a *similarity value* to each entity type in E's hierarchy based on its location in the hierarchy relative to E. The similarity value formalizes some abstract notions such as E is more similar to its siblings than its cousins, more similar to its parent than its grandparent, etc. If using the most familiar mapped model constructs still cannot yield a

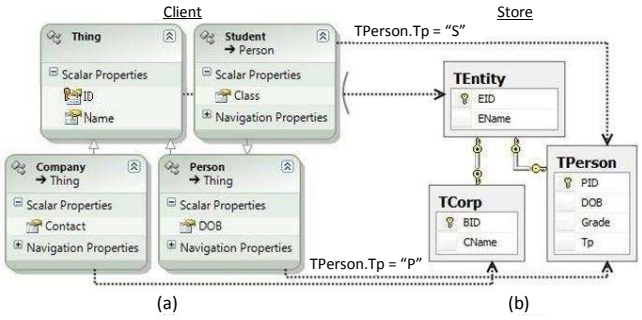

Figure 2: An example pair of client (a) and store (b) models, with a mapping between them. The left "parenthesis" is to indicate that the mapping line from `Thing` goes behind the `Student` entity type

Table 1: The mapping relation for the models and mapping in Figure 2 (column $\mathcal{CC}$ not shown, since the mapping has no client conditions)

CE	CP	ST	SC	SX	K	D
Thing	ID	TEntity	EID	—	Yes	Guid
Thing	Name	TEntity	EName	—	No	Text
Company	ID	TCorp	BID	—	Yes	Guid
Company	Contact	TCorp	CName	—	No	Text
Person	ID	TPerson	PID	Tp=P	Yes	Guid
Person	DOB	TPerson	DOB	Tp=P	No	Date
Student	ID	TPerson	PID	Tp=S	Yes	Guid
Student	DOB	TPerson	DOB	Tp=S	No	Date
Student	Class	TPerson	Grade	Tp=S	No	Text

consistent scheme — for instance, if E has only two siblings, where one is mapped TPC and the other TPT — MoDEF relies on a user-specified default to generate the new store constructs and mapping. We believe this scenario to be rare in practice. The default is also used if the new entity type is the root of a new hierarchy, and thus there is no mapping information to mine.

Internally, MoDEF translates an EF mapping M into a *mapping relation* $\mathcal{M}(\mathcal{CE}, \mathcal{CP}, \mathcal{CX}, \mathcal{ST}, \mathcal{SC}, \mathcal{SX}, \mathcal{K}, \mathcal{D})$ with the following attributes:

- $\mathcal{CE}, \mathcal{CP}, \mathcal{CX}$: Client entity type, property, conditions
- $\mathcal{ST}, \mathcal{SC}, \mathcal{SX}$: Store table, column, conditions
- $\mathcal{K}$: a flag indicating if the property is part of the key
- $\mathcal{D}$: The domain of the property

The mapping relation is a pivoted form of an EF mapping, where each row represents a property-to-property mapping for a given set of conditions. Table 1 shows the mapping relation for the models and mapping in Figure 2.

In the mapping relation, each mapping scheme manifests itself as an invariant. Given a mapping relation $\mathcal{M}$, if the hierarchy H_E for entity type E is mapped using TPT and $\Phi(E)$ is the set of entity types in H_E limited to those that meet a similarity threshold, then the following must hold:

1. All entity types are mapped to distinct tables. For each pair of entity types $E', E'' \in \Phi(E)$, $E' \neq E''$: $\pi_{SE}\sigma_{CE=E'}\mathcal{M} \cap \pi_{SE}\sigma_{CE=E''}\mathcal{M} = \emptyset$
2. Non-key properties are never re-mapped in derived entity types. For each property P in each entity type $E' \in \Phi(E)$: $|\sigma_{\neg K}\sigma_{CP=P}\sigma_{CE=E' \vee CE \text{ inherits from E'}}\mathcal{M}| = 1$

In Table 1, Thing, Company, and Person satisfy the TPT invariants. When a user adds a child entity type to Thing, MoDEF adds it with a TPT-style mapping by adding a dedicated table and not re-mapping inherited properties.

When one adds new client schema constructs, MoDEF issues queries against $\mathcal{M}$ to determine which invariants hold. In essence, it mines an EF mapping in search of mapping schemes. MoDEF supports the TPT and TPC schemes for hierarchy mapping, plus the following:

- Hybrids between TPT and TPC
- TPH, where all properties for all entity types map to distinct columns
- TPH, where properties may share a column if they have the same name and type (this scheme is how Ruby on Rails maps hierarchies)
- TPH, where properties may share a column if they have the same type (minimizing column count)
- Any of the above TPH schemes, where there is no discriminator column (type membership is determined by the presence or absence of values)
- Consistent horizontal partitioning, i.e., all entity types are partitioned on a particular property's value
- Patterns of store-side conditions on mapping fragments, which can be used to assign constant values on insert

Handling renamed or dropped constructs is much simpler than handling added constructs. When an entity type, property, or association is dropped, MoDEF removes all rows in the mapping relation that refer to the dropped object, and any store objects that become unmapped. Renames propagate through the mapping any time the renamed client construct and its mapped store construct have the same name.

MoDEF supports refactoring an association into an inheritance relationship by treating the association as if it were a TPT relationship that has not been identified yet. A 1-1 association along a foreign key is exactly how the TPT scheme represents inheritance in a relational database. Thus, the refactoring leaves the store unchanged and alters the mapping to institute the inheritance on the client side.

MoDEF also supports refactoring a client model by moving a property of an entity type to that entity type's parent or to one of its children. Movement of a property from an entity type to its parent is lossless, whereas moving the property to a child will delete any data in that property for instances that do not belong to the new entity type. For TPT, MoDEF adds a new column to the destination table (corresponding to the property's destination entity type), moves all applicable data across the foreign key linking the two tables, and drops the old column. For TPC and TPH, no data movement or column modification is necessary, though individual data values may be deleted if no longer applicable or set to a default value if the property is non-nullable and instances already exist of the destination entity type.

4. EXAMPLE EVOLUTION

When one selects the "Generate database from model" command in MoDEF, the resulting SQL script is punctuated by comments that describe the client-side action that

```
-- Add property 'CEO' to entity type 'Company'
ALTER TABLE [TCorp] ADD COLUMN [CEO] varchar(40);

-- Add property 'Major' to entity type 'Student'
   with inherited value 'General Studies'
ALTER TABLE [TPerson]
   ADD COLUMN [Major] varchar(50);
UPDATE [TPerson] SET [Major] = 'General Studies'
   WHERE [Tp] = 'S';

-- Rename property 'DOB' of entity type 'Person'
   to 'BDay'
EXEC sp_rename 'TPerson.DOB', 'BDay', 'COLUMN';

-- Move property 'Contact' of entity type
   'Company' to entity 'Thing'
ALTER TABLE [TEntity]
   ADD COLUMN [CName] varchar(50);
UPDATE [TEntity] SET [CName] = child.[CName]
   FROM [TEntity] as parent
   LEFT OUTER JOIN [TCorp] as child
                   ON parent.EID = child.CID
ALTER TABLE [TCorp] DROP COLUMN [CName];
```

Figure 3: An example of punctuated SQL, the result of incremental updates to the model in Figure 2

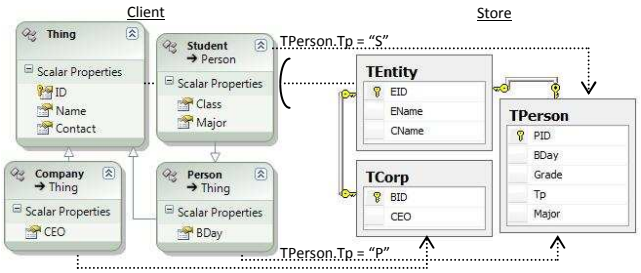

Figure 4: The client and store models after the changes in Figure 3

generated a given fragment of the script. Figure 3 shows the result of doing four incremental changes to the client model shown in Figure 2: two added properties (one with an inherited value), a renamed property, and a moved property. After these changes have been made, the client and store models are as shown in Figure 4, and the mapping is as shown in Table 2.

5. RELATED WORK AND FUTURE WORK

A wealth of research has been done on schema evolution [7], but very little has been done on co-evolution of schemas connected by a mapping. One prominent example is MeDEA, which uses manual specifications of update policies [1]. For each change to the client schema and for each mapping, one can manually specify policies that prescribe the effect on the store. The advantage that MoDEF provides is that one does not need to specify any policies manually, other than a single global default scheme; the developer uses the existing tools provided with Visual Studio in the normal way.

Table 2: The mapping relation from Table 1 after applying the changes from Figure 3

CE	CP	SE	SP	SC	K	D
Thing	ID	TEntity	EID	—	Yes	Guid
Thing	Name	TEntity	EName	—	No	Text
Thing	Contact	TEntity	CName	—	No	Text
Company	ID	TCorp	BID	—	Yes	Guid
Company	CEO	TCorp	CEO	—	No	Text
Person	ID	TPerson	PID	Tp=P	Yes	Guid
Person	BDay	TPerson	BDay	Tp=P	No	Date
Student	ID	TPerson	PID	Tp=S	Yes	Guid
Student	BDay	TPerson	BDay	Tp=S	No	Date
Student	Class	TPerson	Grade	Tp=S	No	Text
Student	Major	TPerson	Major	Tp=S	No	Text

The Both-As-View (BAV) federated database language can express non-trivial mappings between schemas, though it does not handle inheritance [4]. For some schema changes (to either schema in BAV), either the mapping or the other schema can be adjusted to maintain validity. Many cases require manual intervention for non-trivial mappings.

Two cases of schema evolution have been considered in data exchange, one on incremental client model changes [8], and one where evolution is represented as a mapping [9]. Both cases focus on "healing" the mapping between schemas, leaving the non-evolved schema invariant. New client constructs do not translate to new store constructs, but rather add quantifiers or Skolem functions to the mapping, which means client constructs do not receive any persistence.

A prominent feature of EF is that it compiles mapping fragments into views that describe how to translate data from a store model into a client model and vice versa. An active area of our research is to translate incremental changes to a model into incremental changes to these compiled views.

6. REFERENCES

[1] E. Domínguez, J. Lloret, A. L. Rubio, and M. A. Zapata. Evolving the Implementation of ISA Relationships in EER Schemas. *ER Workshops 2006*, LNCS 4231.

[2] Hibernate. Available at http://www.hibernate.org/.

[3] A. Malpani, P. A. Bernstein, S. Melnik, and J. F. Terwilliger. Reverse Engineering Models from Databases to Bootstrap Application Development. *ICDE 2010*.

[4] P. McBrien and A. Poulovassilis. Schema Evolution in Heterogeneous Database Architectures, a Schema Transformation Approach. *CAiSE 2002*.

[5] S. Melnik, A. Adya, and P. A. Bernstein. Compiling Mappings to Bridge Applications and Databases. *ACM TODS* 33(4) (2008).

[6] Ruby on Rails. http://rubyonrails.org/.

[7] E. Rahm and P. A. Bernstein. An Online Bibliography on Schema Evolution. *SIGMOD Record* 35(4), 2006.

[8] Y. Velegrakis, R. J. Miller, and L. Popa. Preserving Mapping Consistency Under Schema Changes. *VLDB Journal*, 2004, 13(3).

[9] C. Yu and L. Popa. Semantic Adaptation of Schema Mappings When Schemas Evolve. *VLDB 2005*.