Mutual Satellites: Round-trip Modeling for Complete Applications

Lars Thorup, Sune Gynthersen, Kristian Dupont
Mutual Satellites A/S
Nørre Søgade 27A stv
1370 Copenhagen K, Denmark
+45 7026 2118

{lars.thorup, sune.gynthersen, kristian.dupont}@mutual-satellites.com

Abstract

During the presentation we will demonstrate a new modeling tool, implemented as an Add-In for Microsoft Visual Studio. Our tool provides complete round-trip capabilities enabling application manipulation using either the model or the source code. The tool manipulates all three application layers of conventional enterprise applications, currently with support for SQL, C# and XAML.

Categories and Subject Descriptors D.2.3 [**Software Engineering**]: Coding Tools and Techniques – *Object-oriented programming*

General Terms Design

Keywords Model Driven Development, UML, Entity Relationship, C#, SQL, XAML, round-trip

1. Introduction

Mutual Satellites is a new kind of tool for Model-Driven Development[1]. Mutual Satellites manipulates all three application layers: the database layer, the business object layer and the user interface layer. The software developer retains complete control over the details of the source code because Mutual Satellites implements complete round-trip engineering on all application layers, see Figure 1.

By achieving this, we improve on existing modeling tools. Some tools (typically generation-based) succeed in generating complete applications but then lack round-trip capabilities. Other tools (typically UML and database modeling) succeed in providing round-trip capabilities but then lack support for more than a single application layer.

Mutual Satellites supports a modeling language based on classic data modeling diagrams, like UML[2] or Entity-Relationship[3], extended with data validation and other presentation-level features. The tool is currently being implemented for Microsoft Visual Studio 2005 and targets enterprise applications built with SQL, C# and XAML for the respective three application layers. The tool fits with the existing designers in Visual Studio.

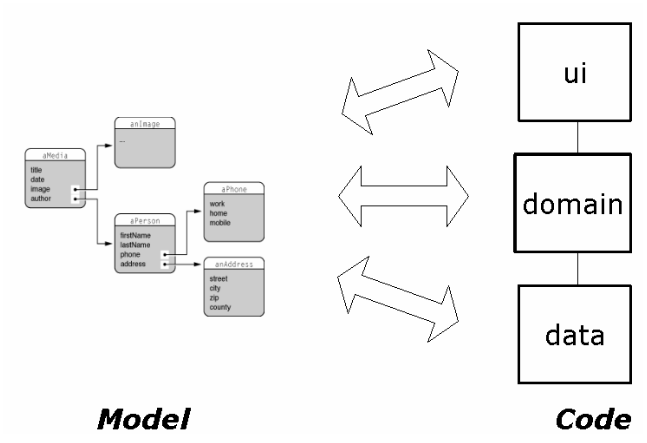


Figure 1: Round-trip modeling for three layers

2. Architecture

Mutual Satellites works by annotating the source code with references to the model. This means that the source code itself contains the model, thus there is no separate representation of the model. From the annotations the tool can infer a graphical representation of the model, allowing the developers to work directly on this model and having the tool convert all model changes back into corresponding source code changes. Similarly the developers can make any kind of changes to the source code and then have the tool re-infer the resulting model. Therefore the tool has full round-trip capabilities while still giving developers complete control of all the details of the source code.

Annotations can be potentially added to source code in any programming language and thereby make connections from different layers to a common model. In the case that Mutual Satellites contains language packs for all programming languages used in an application, the tool will allow developers to maintain a model for the complete application. Mutual Satellites works with existing programming languages, thus allowing software developers to leverage their existing competences.

OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

3. Benefits

In *The Pragmatics of Model-Driven Development*[4], Bran Selic, goes through a list of seven challenges that MDD tools must address before they will be successful in the market. Mutual Satellites addresses all seven challenges:

3.1 Model/code correspondance

With Mutual Satellites, model-level observability is addressed with a feature we call "Go to model". After an error is detected by the compiler or the run-time system, the developer can go directly from the location of the error in the source code to the corresponding location in the model.

3.2 Version control

With Mutual Satellites, the model is completely represented by the annotations in the source code, so all merging and diff'ing of model changes are handled at the source code level, allowing the use of existing version control tools.

3.3 Model executability

With Mutual Satellites, the model is completely represented by annotated code, and this code can in general be compiled and run at any time, effectively executing the model.

3.4 Efficiency of generated code

With Mutual Satellites, the source code can be freely adapted and optimized by the developer for “occasional critical cases” without influencing the model for the remaining code.

3.5 Scalability

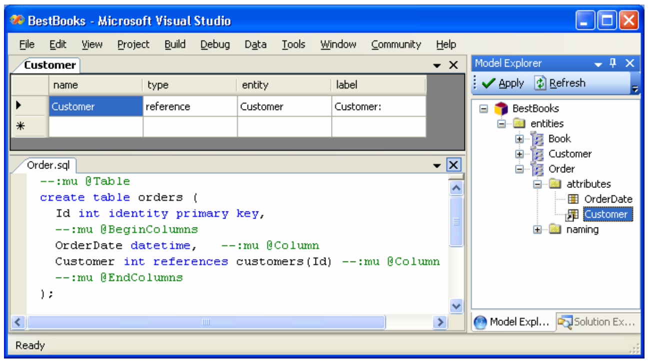
With Mutual Satellites, changes to the model are synchronized to the corresponding changes in the source code by modifying only the parts of the source code that have annotations referencing the affected parts of the model, thereby ensuring scalability to very large code bases.

3.6 Integration with legacy development environments

In essence, the purpose of Mutual Satellites is to guide the developer in editing the code, and Mutual Satellites is not dependent on any of the other tools used by the developer. The developer can utilize any tools best suited for the purpose. For example, the visual designers of Visual Studio can be used to change the visual appearance of the user interface without disturbing the use of Mutual Satellites to change the logical structure of the user interface.

3.7 Integration with legacy systems

With Mutual Satellites, developers can freely adapt the source code and insert calls to any existing legacy code libraries and other legacy software used by the project.



	name	type	entity	label
▶	Customer	reference	Customer	Customer:
*				

```
--:mu @Table
create table orders (
  Id int identity primary key,
  --:mu @BeginColumns
  OrderDate datetime,   --:mu @Column
  Customer int references customers(Id) --:mu @Column
  --:mu @EndColumns
);
```

Figure 2: Mutual Satellites

4. Demonstration Overview

Mutual Satellites is currently in the early stages of development. Figure 2 shows how model and code is currently displayed inside Visual Studio. The model is shown as a tree structure on the right and a property page on the top, while the editor at the bottom shows some of the annotated source code.

During the demonstration we will show two important scenarios of model driven development:

- Using the model, we will add a new data item attribute to the application and see that the new attribute is then fully supported through all layers.
- Using the source code editor, we will fine-tune some details of the code and see that these changes are not affected by subsequent changes made via the model.

5. References

[1] Steimann, F., Kühne, T. Coding for the Code. *ACM Queue* vol. 3, no. 10 - December 2005

[2] OMG, UML 2.0, http://www.uml.org

[3] Chen, P. P. The entity-relationship model – toward a unified view of data. *ACM Trans. Database Systems* 1(1):9-36, 1976.

[4] Selic, B. The Pragmatics of Model-Driven Development. *IEEE Software, 2003/05 (September/October 2003)*, 19-25.