Aufgabenstellung

Einführung

Am .NET CC wurden im Rahmen des Softnet-Projektes Architekturvarianten für Enterprise Applikationen mit .NET untersucht und mit verschiedenen Referenzimplementationen für eine einfache Bestellapplikation konkretisiert. Eine Variante dieser Bestellapplikation (HsrOrderApp_S3) verwendet im BusinenssLayer ein objektorientiertes Domain-Modell. Dieses Modell wird mit einem ojektrelationalen Mapping auf die DB-Tabellen abgebildet und bietet ein sehr grob granulares und doch mächtiges Webservice Interface an. Verglichen mit der Funktionalität, welche die Applikation anbietet, wurde sehr viel Code benötigt. Dafür ist das resultierende System sehr flexibel, überschaubar und erweiterbar.

Zielsetzungen dieser Diplomarbeit

Die erste Aufgabe der Diplomanden ist die Analyse der bestehenden Applikation. Es sollen Implementationen gesucht werden, wo mit generativen oder generischen Ansätzen die Komplexität der SW und damit die Entwicklungs- und Wartungsaufwendungen reduziert werden können. Die Resultate müssen von allgemeiner Natur sein und sollen auch in anderen Enterprise Applikationen einsetzbar sein. Nachfolgend wird kurz auf einige bereits identifizierte Problemkreise eingegangen:

Objektrelational Mapping

Die HsrOrderApp_S3 arbeitet mit einem Domain-Model in klassischem OO Stil. Das Mapping dieses Modells benötigte einiges an Codieraufwand. Es wurden aber bereits generische Ansätze verwendet um diesen in Grenzen zu halten (siehe z. B. AbstractMapper).

Modell Transformationen

Die HsrOrderApp_S3 benötigt eine zweistufige Modelltransformation, wenn sie von einem Web-service aus angesprochen wird: Eine zwischen der Fassade und dem Domain-Model und eine zweite für den Webservice. Diese Transformationen sind für den Entwickler sehr mühsam und fehleranfällig.

Ein möglicher Lösungsansatz ist die Verwendung des Tools CodeSmith. CodeSmith ist ein sehr mächtiges OpenSource Werkzeug für das automatische Generieren von Code. Für die Parametrisierung der CodeSmith Templates soll XML verwendet werden.

Erwartete Resultate

1. Analyse und Dokumentation der dieser Arbeit zugrundeliegenden Patterns und "Best Practices"für den Entwurf des OO-Domain-Modell, der Business-Facades und der Service-Interfaces.

2. Analyse, wo die Verwendung von generischen oder von generativen Ansätzen sinnvoll ist. Ausarbeitung von "Best Practices"(inkl. Varianten, untermauert mit Code-Beispielen)

3. Entwicklungsframework für generative Ansätze für Modell-Transformationen basierend auf CodeSmith:

    - Eine Beschreibungssprache basierend auf XML und XML-Schema für die Parametrisierung der CodeSmith Templates.
    - Ein Satz von Templates für Code Smith die unter anderem das O/R Mapping und die Modell Transformationen generativ lösen.
    - Einbindung der Templates in den Make-Prozess (für Visual Studio .NET).

- Evtl. ein Tutorial zu Codesmith.

4. Überarbeitete und dokumentierte HsrOrderApp
5. allfällige weitere Aspekte und Konzepte, die sich im Laufe der Arbeit ergeben.

Vorgehensweise (Vorschlag):

1. Die HsrOrderApp_S3 als ganzes, die verwendeten Architektur- und Designpatterns und das O/R Mapping müssen von den Diplomanden studiert und verstanden werden. Litera-turstudium (u.a das Buch von Fowler, [1]).
2. Technologiestudien: CodeSmith, notwendige .NET-Technologien.
3. Analyse wo die Verwendung von einem generischen Ansatz sinnvoll ist.
4. Eine Beschreibungssprache basierend auf XML und XML-Schema die als Parametrisierung der CodeSmith Templates dient definieren.
5. Templates erstellen.
6. Implementation der HsrOrderApp basierend auf den erarbeiteten generativen und generischen Ansätzen. Dokumentation.

Entwicklungsumgebung

MS Windows 2000 mit IIS, MS SQL Server 2000, Visual Studio :NET 2003

Dokumentation

Es ist nach einem Projektplan zu arbeiten. Der tatsächliche Aufwand ist zu erfassen. Ein individuelles Projekttagebuch ist zu führen, aus dem ersichtlich wird, welche Arbeiten durchgeführt wurden (inkl. Zeitaufwand). Diese Angaben sollten u.a. eine individuelle Beurteilung ermöglichen.

Abzugeben ist ein Bericht (1 Exemplar auf Papier, 3 Exemplaren auf CD-ROM), welcher enthält:

- die Aufgabestellung
- eine Zusammenfasssung der Arbeit auf einer Seite auf dem dafür vorgegebenen Formular
- ein Management Summary
- ein in sich geschlossener technischer Bericht mit einer ausgiebigen Diskussion und Dokumentation der geforderten Problemanalysen und der Problemlösungskonzepte inklusive Literaturstudium (gemäss Kapitel)
- persönliche Berichte der Studierenden zu ihren Erfahrungen in der Arbeit
- bei der Arbeit erstellten Dokumente
- die Programme (Softcopy)
- der Projektplan mit den geplanten und tatsächlichen Aufwände, Projekttagebuch

Aus dem Bericht muss klar hervorgehen, wer für welchen Teil der Arbeit und des Berichtes verantwortlich ist.

Nach Abgabe des Berichts ist ein Poster vorzubereiten, welches die wesentlichen Informationen über die Arbeit für die Ausstellung der Diplomarbeit enthält.

Termine

Abgabe der Aufgabestellung und Beginn der Arbeit: Montag 20. Okt. 03
Zwischenpräsentation für Experte: Mi. 26. Nov, nachmittags
Abgabe des Berichts und Ende der Arbeit: Mi. 10. Dez. 03, 17 Uhr (prov.)
Ausstellung und Vorführung der Arbeit: Fr. 12. Dez.03
Mündliche Prüfung zur Diplomarbeit: 12.01.03 - 17.01.03
Ausstellung der Poster und Diplomfeier: Freitag 23.01.2004

In der ersten Woche sind Projektauftrag und die Projekt-Planung mit Meilensteinen zu erstellen. Beides muss vom Auftraggeber und Betreuer genehmigt werden. Bei der Projektplanung ist zu beachten, dass bis zur Zwischenpräsentation für den Experten erste konkrete Resultate vorliegen.

Betreuer

Betreuer: H. Huser
Experte: Dr. Hans Bärfuss

Mit dem Betreuer ist (mindestens) eine wöchentliche Besprechung durchzuführen. Termin:

Falls notwendig, können weitere Besprechungen / Diskussionen einberufen werden. Themen:

- Stand des Projektes
- Nächste Schritte
- Überprüfung der Meilensteine
- Diskussion fachlicher Themen (bei Bedarf)

Zu jeder Besprechung muss ein Kurzprotokoll erstellt werden, welches spätestens 2 Tage nach der Sitzung per e-mail an den Betreuer zu senden ist.

Rapperswil, 20. Okt. 03, H. Huser

Abstract

STORM ist ein Framework das es ermöglicht, Code für das O/R Mapping zu generieren. Das Framework besteht einerseits aus generischen Klassen, die von den selbst geschriebenen Klassen benutzt werden können und andererseits aus Templates, die benutzerdefinierte Klassen erzeugen. Die Kernidee von *STORM* ist, dass der Entwickler abstrakte Klassen schreibt, in denen er das Mapping zwischen Klassen der Applikation und Tabellen der Datenbank angeben kann. Diese abstrakten Klassen können anhand von Attributen, die in *STORM* definiert wurden, parametriert werden. Diese Attribute und deren Werte werden zur Zeit der Code Generierung gelesen und dienen als Input der Templates. Der Output dieser Templates sind die auf das Problem angepassten Domain Objekt und Mapper Implementationen.

Der generierte Code übernimmt Aufgaben wie Lazy Load, Optimistic Offline Locking, Verwaltung und Ausführung von Insert, Update und Delete sowie das dynamische Erstellen von Abfragen mittels eines Query Objects. Zudem stellt er eine Factory zur Verfügung, um Instanzen der generierten Klassen erstellen zu können.

Um den Code zu generieren wurde das Open Source Tool CodeSmith eingesetzt.

Management Summary

Die Problematik

Für die Entwicklung von Applikationen wird meist eine objektorientierte Sprache wie Java oder C# verwendet. Dabei sind in einem Objekt alle Daten und Funktionen, die für eine bestimmte Aufgabe notwendig sind, zusammengefasst. Dies erlaubt es, reelle Objekte und deren Abhängigkeiten einfach und strukturiert in der Software abzubilden.

Bei Datenbanken hingegen sind die Strukturelemente keine Objekte, sondern Tabellen, die die durch Relationen miteinander verknüpft sind. Dieser Unterschied wird als “type mismatch” bezeichnet.

Da viele Applikationen Datenbanken verwenden, muss eine Möglichkeit gefunden werden, diese Strukturen zu mappen. Dieses Problem ist schematisch in Abbildung 1 dargestellt.

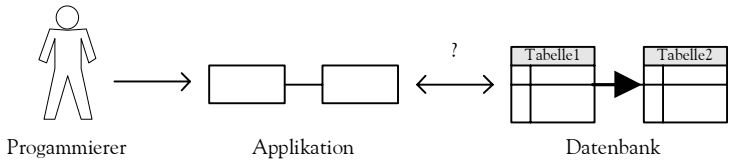


Abbildung 1: Verschiedene Strukturen bei der Applikation und der Datenbank.

Die Applikation

Der Name der Applikation ist *S*torm *T*emplate-based *O*bject *R*elational *M*apper (*STORM*). Das Ziel von *STORM* war es, ein Framework zu erstellen, das den Code für das erwähnte Mapping generiert. Damit wird dem Entwickler ein Teil seiner Arbeit abgenommen.

Die Idee ist, dass der Entwickler abstrakte Domain Objekte implementiert, welche er mit Attributen parametrieren kann. Durch die Attributierung kann er einerseits das Domain Modell und andererseits das Datenbank Modell vorgeben. Anhand dieser Angaben wird anschliessend der gesamte Mapping Code generiert. Dieser Vorgang ist schematisch in Figur 2 dargestellt.

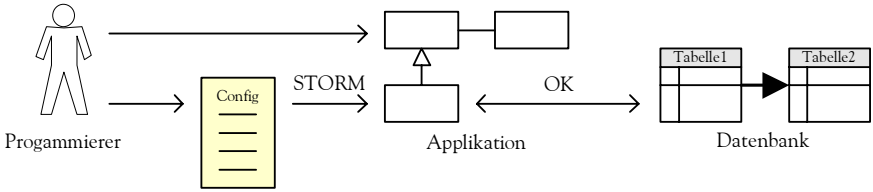


Abbildung 2: Der Code für das Mapping wird von *STORM* erstellt.

Die Vorteile

Die folgende Liste gibt einen Überblick der Vorteile von *STORM*.

- Die Entwicklungszeit einer Applikation kann stark minimiert werden.
- Der generierte Mapper Code ist fehlerfrei.
- Der Code wird durch die Verwendung von Templates besser wartbar.
- Die Konfiguration wird an einem Ort, nämlich den abstrakten Domain Objekten, vorgenommen.
- Der Entwickler muss sich weniger mit der Datenbank auseinandersetzen.
- Der Generierungsprozess ist im Visual Studio .NET eingebunden.

Ausblick

Verschiedene Dinge konnten während dieser Arbeit nicht umgesetzt werden, da die Zeit sehr begrenzt war. Daraus ergibt sich folgende Liste von Erweiterungsvorschlägen.

- Meherere Locking Mechanismen werden unterstützt.
- Alle möglichen Mapping Typen einer Datenbank werden unterstützt.
- Zusätzlich zum Mapping zwischen Domain Objekten und Datenbank wird auch das Mapping zwischen Web Service Facade und Domain Objekten unterstützt.
- Als Konfigurationsmöglichkeit werden nicht nur abstrakte Klassen angeboten, sondern auch eine Datenbankverbindung, aus der direkt die Domain Klassen erzeugt werden.

Es wäre denkbar, die oben genannten Punkte in einer weiteren Arbeit zu implementieren. Ebenso müssten noch mehrere Testapplikationen mit *STORM* entwickelt werden, um das Framework besser testen zu können.

Die Gefahr bei der Erweiterung der jetzt bestehenden Templates ist, dass sie sehr unübersichtlich werden. Man müsste den Aufbau der Templates allenfalls überarbeiten und genau planen, wie dieser aussehen müsste, damit der Code sauber und übersichtlich wird. Eine Lösung könnte zum Beispiel sein, ein Template in mehrere Templates aufzuteilen.

Der Lösungsansatz mit den attributierten, abstrakten Klassen erscheint uns sinnvoll.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I

Anforderungsspezifikation

Kapitel 1

Vision

1.1 Ziel der Arbeit

Bei den meisten Applikationen gibt es eine bestimmte Anzahl von Codezeilen, die repetitiven Code darstellen. Das bedeutet, dieser Code muss bei jeder Applikation neu geschrieben werden. Ansätze diese Arbeit auf ein Minimum zu reduzieren sind generative bzw. generische Programmiertechniken.

Diese Arbeit soll ein Framework zur Verfügung stellen, mit dem repetitiver Code generiert werden kann. Damit wird erreicht, dass die Software besser zu Warten und der Entwicklungsprozess optimiert werden kann. Die Erkenntnis aus dieser Arbeit soll aus dem Bericht hervorgehen und soll zeigen, wie sinnvoll ein generativer Ansatz ist, um einen Objekt-Relationalen Mapper zu entwickeln.

Das entwickelte Framework soll möglichst einfach zu benutzen sein. Ein wichtiger Aspekt der Arbeit ist dabei die Erarbeitung eines geeigneten Konzeptes für die Parametrierung der Code Generierung, damit diese möglichst generell eingesetzt werden kann.

1.2 Positionierung

Kapitel 2

Detaillierte Anforderungen

Im Folgenden werden die Anforderungen an die Software definiert. Nicht definiert werden hier die übrigen Ziele, die in der Vision (1.1) genannt wurden.

2.0.1 Muss-Ziele

1. Eine Lösung für die Konfiguration der Code Generierung muss gefunden werden.

2. Benötigte Templates für das Tool CodeSmith [5] werden erstellt.

3. Eine Lösung für die Integration in die Visual Studio .NET Umgebung wird gefunden.

4. Falls benötigt, wird ein Plugin für CodeSmith geschrieben.

5. Eine überarbeitete Version der HsrOrderApp_S3 wird erstellt (mit den erstellten CodeSmith Templates) und dokumentiert. Daraus entsteht die HsrOrderApp_S4.

2.0.2 Optionale Ziele

1. Eine zusätzliche Beispielapplikation zur Veranschaulichung des Nutzen der Arbeit.

2. Ein Wizard für das Visual Studio .NET, um das Erstellen der Konfigurations-Dateien zu erleichtern.

Da bei dieser Arbeit ein entscheidender Teil ist, ein Lösung für ein Problem zu finden, können auch keine spezifischen, sondern nur allgemein formulierte Anforderungen aufgeführt werden. Das Problem ist, dass wir noch nicht wissen, wie diese Lösung aussehen wird und deshalb auch keine Anforderungen an diese Lösung stellen können. Dies entspricht auch der Aufgabenstellung.

Teil II

Analyse

Der folgende Teil enthält Kapitel in Deutsch und Englisch. Wir haben entschieden, nur den technischen Bericht in Englisch zu verfassen, hatten zu diesem Zeitpunkt jedoch schon einzelne Kapitel geschrieben.

Chapter 3

Fundamental approach

Object-relation Mapping can be done in many ways. First, of course, one can implement the whole code by hand. As already mentioned, this is not the scope of this thesis and is therefore not a possible solution. Second, we have a mechanism to automate the coding part. In this thesis, we try to find a good solution to generate the code. Another approach would be to supply a set of generic classes, which is not our aim, either.

To generate as much code as possible, we need to have information about the database structure as well as about the domain model and how to map these two. This can be quite a lot the programmer needs to know but usually should not be a problem. The intend of *STORM* is to provide a programmer a solution, which allows him to specify all these information in a, for a programmer, natural way.

The following sections describe 3 possible ways to implement such a tool as *STORM*. The pros and cons are mentioned. There is no doubt that there exist more possibilities but we hope to give a reasonable overview with the following selection.

3.1 XML Schema

Figure 3.1 shows a first possible mechanism to describe the information needed to generate mapping code. On the right, an XML Schema file is used to reflect the database structure. A sample of such a file is shown in

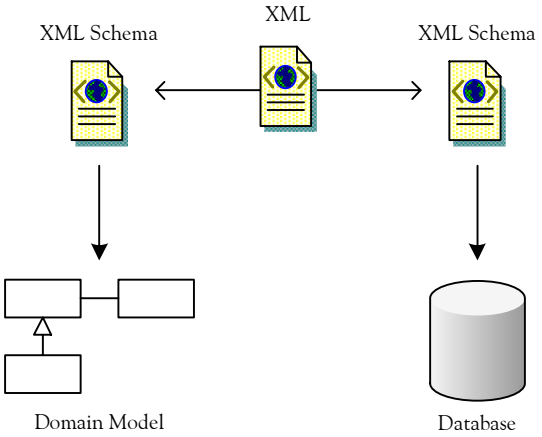


Figure 3.1: Mapping description with XML Schema

Listing 3.1. This is probably not the most sophisticated XML schema possible but nevertheless it is possible to describe a database structure with such a file. We implemented a XML schema parser which read the content of this file and used it as input for a CodeSmith template.

The main problem with this approach is the XML schema language. It is not possible to describe all information needed to generate the mapping code in a schema file. This limitation results in the fact that another XML schema file is needed to describe the domain model structure (as shown on the left in Figure 3.1). Additionally an XML file is needed to specify the mapping between the two schemes. For example, this XML file would include a tag describing which database table matches to which class in the domain model.

There are a few drawbacks with this solution. One of those is that an XML schema is not a natural way a programmer would describe a database structure nor a domain model. Another drawback is its implementation. The schema description must undergo certain restrictions for consistency. A programmer therefore has to adopt and respect this restriction when writing a schema file. The implementation itself is not easy as well, because a mapping strategy needs to be defined. To define such a strategy which allows to specify every possible mapping is not an easy task.

Listing 3.1: XML schema describes database structure

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="BusinessObjects1"
    targetNamespace="http://tempuri.org/BusinessObjects1.xsd"
  elementFormDefault="qualified"
      xmlns="http://tempuri.org/BusinessObjects1.xsd"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:group name="classes">
    <xs:sequence>
      <xs:element name="Person" type="PersonType" />
      <xs:element name="Address" type="AddressType" />
    </xs:sequence>
  </xs:group>
  <xs:complexType name="AddressType">
    <xs:sequence>
      <xs:element name="Person" type="PersonType" />
      <xs:element name="Street">
        <xs:simpleType>
          <xs:restriction base="xs:string">
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="City">
        <xs:simpleType>
          <xs:restriction base="xs:string">
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
  </xs:complexType>
</xs:schema>
```

On the other hand, there are benefits which makes this approach worth looking at. XML schema and XML are well defined and standardised. This makes it possible to delegate the task of writing the XML files to a tool such as a database design tool. The only thing which cannot be automated is the XML file describing the mapping.

3.2 Abstract Class with Attributes

A second approach to describe the mapping is to write an abstract class with custom attributes. A code snippet of such a class is shown in Listing 3.2. This class file is compiled at runtime and the custom attributes can be read using reflection from the resulting dll. With these attributes, all needed information can be retrieved.

Listing 3.2: Abstract class with custom attributes

```
[Table("Addresses", true),
VersionField("chTimeStamp", typeof(Timestamp))]
public abstract class Address : DomainObject
{
  [Column("AddressID"),
    PrimaryKey]
  public abstract int AddressId {get; set;}

  [Column("City")]
  public abstract string City {get; set;}

  [Column("Street")]
  public abstract string Street {get; set;}
}
```

The main advantage of this is that a programmer can define everything in a single file rather than multiple files. Furthermore it is a more natural way for programmers because they are used to write classes.

One could say that a drawback of attributes is that they cannot be changed at runtime. This is true but also, it is not required. Every information given in this abstract class is static within the scope of an application. As mentioned above, C# reflection mechanism is used to read the settings. This is constricted to the process of generating code. The generated code does not make use of reflection. This would be possible but would result in a poorer performance.

From our point of view, this approach is the most feasible.

3.3 Extended C# Compiler

This is the most advanced approach and would give the highest possible flexibility. The basic idea is to take an existing C# compiler and extend it to work with custom defined keywords. With this new introduced keywords, the mapping information could be given in a well defined, familiar manner.

Due to the limited time for this thesis, we do not follow this approach in more detail, although it would be very interesting and promising. A possible solution could be to use a compiler generation tool such as Coco/R [3].

Kapitel 4

Konzeptionelles Modell

4.1 Generisch vs. generativ

Zuerst muss entschieden werden, welche Teile des Codes generiert werden können und bei welchen Teilen ein generativer Ansatz besser ist. Um mit der HsrOrderApp_S3 als Vorlage möglichst bald eine lauffähige HsrOrderApp_S4 mit generiertem Code zu erhalten, werden wir uns auf die Generierung von Domain Objekt Klassen und Mapper Klassen beschränken.

Die Beschreibungen für die Generierung des Codes werden in abstrakten Klassen gemacht.

Da UnitOfWork, Identity Map und einige Finder Methoden noch generisch implementiert werden, ist es nötig, dass die Domain Objekte von einer gemeinsamen Oberklasse abgeleitet werden. Die Klasse `DomainObject` ist Teil der *STORM* Library. Die abstrakten Klassen müssen von dieser abgeleitet werden.

Die Abbildung 4.1 zeigt die Vererbungsstruktur vom einem `DomainObject` bis zur generierten Implementation.

Ein Problem, das gelöst werden muss, ist das Arbeiten mit den generierten Implementationen. Weder aus dem selber geschriebenen Code noch aus anderen generierten Klassen, darf auf generierte Klassen zugegriffen werden. Denn der Code muss auch kompilierbar sein, wenn noch kein oder nur ein Teil des Codes generiert wurde.

Damit jedoch der generierte Code auch genutzt werden kann, muss dieser

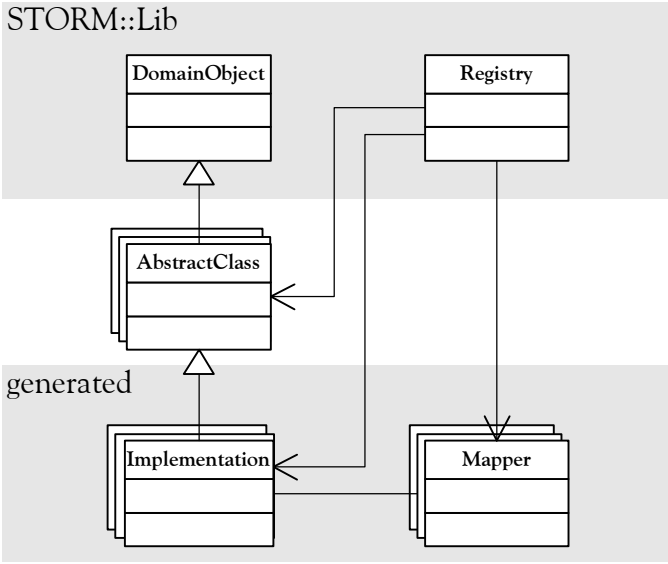


Abbildung 4.1: Vererbungsstruktur mit STORM

an einer zentralen Stelle verwaltet werden. Hier kommt die Registry zum Einsatz.

4.2 Registry

Die generierten Klassen werden mit Attributen versehen. Wenn die Applikation gestartet wird, muss eine Init Funktion der Registry aufgerufen werden, die den gesamten Code per Reflection analysiert und die Implementationen der Domain Objekte sowie die dazugehörigen Mapper sucht und in einer Tabelle ablegt.

Die nachfolgend beschriebenen Factory und Finder Klassen können von der Registry über ein statisches Attribut in der Implementation geholt werden.

4.3 Factory

Soll zum Beispiel ein Objekt der Implementation `PersonImpl` erstellt werden, muss zuerst von der Registry mit dem abstrakten Typ `Person` dessen Factory geholt werden. Wie in Abbildung 4.2 gezeigt, ist die Factory eine interne Klasse der Implementation. Sie kann somit den Implementations Code instanzieren, da er auf jeden Fall existiert.

Die generierte Factory stellt Standard Methoden zur Verfügung. Diese werden aus dem selber geschriebenen Code über das Interface IFactory aufgerufen.

In der abstrakten Klasse können bei Bedarf zusätzliche Factory Methoden beschrieben werden, die im generierten Code implementiert werden. Die Beschreibung geschieht mit Attributen.

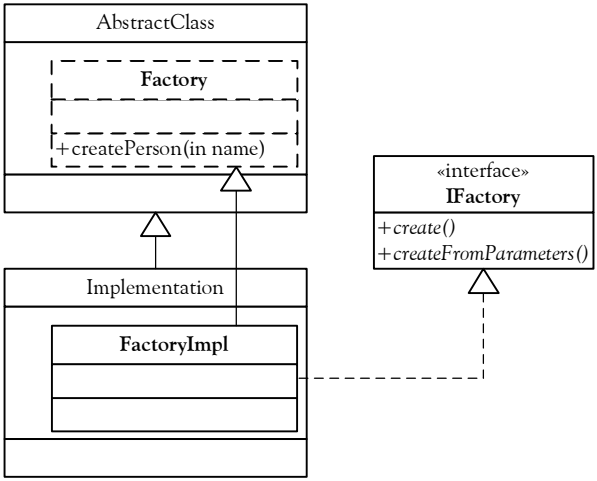


Abbildung 4.2: Business Objekt mit interner Factory Klasse

4.4 Finder

Will man ein Objekt in der Datenbank suchen, muss man auf sogenannte Finder Methoden zugreifen können. Da diese Funktionen generiert werden,

stehen wir vor dem gleichen Problem wie bei der Factory. Wie man in Abbildung 4.3 sehen kann, werden die Finder Klassen ebenfalls als interne Klassen realisiert. Die drei Standard Finder Methoden können über das Interface IFinder aufgerufen werden.

In der abstrakten Klasse können wie bei der Factory zusätzliche Finder Methoden beschrieben werden.

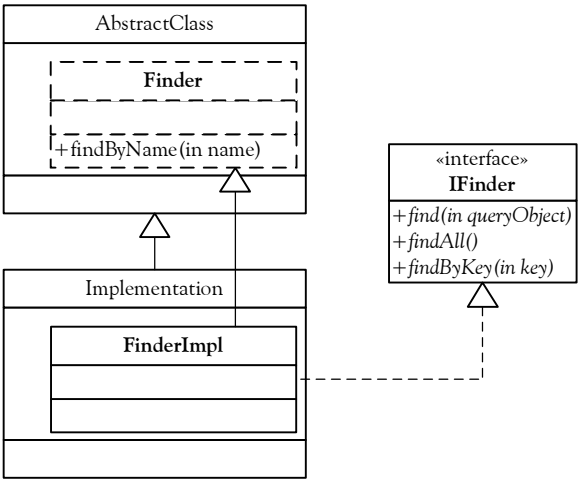


Abbildung 4.3: Business Objekt mit interner Finder Klasse

Die Methode `findAll()` gibt eine Liste mit allen Objekten von einem bestimmten Typ zurück. `findByKey()` sucht ein Objekt mit dem gewünschten eindeutigen Schlüssel. Der Methode `find()` kann ein QueryObject mitgegeben werden, das Bedingungen für die Suche auf der Datenbanken speichert.

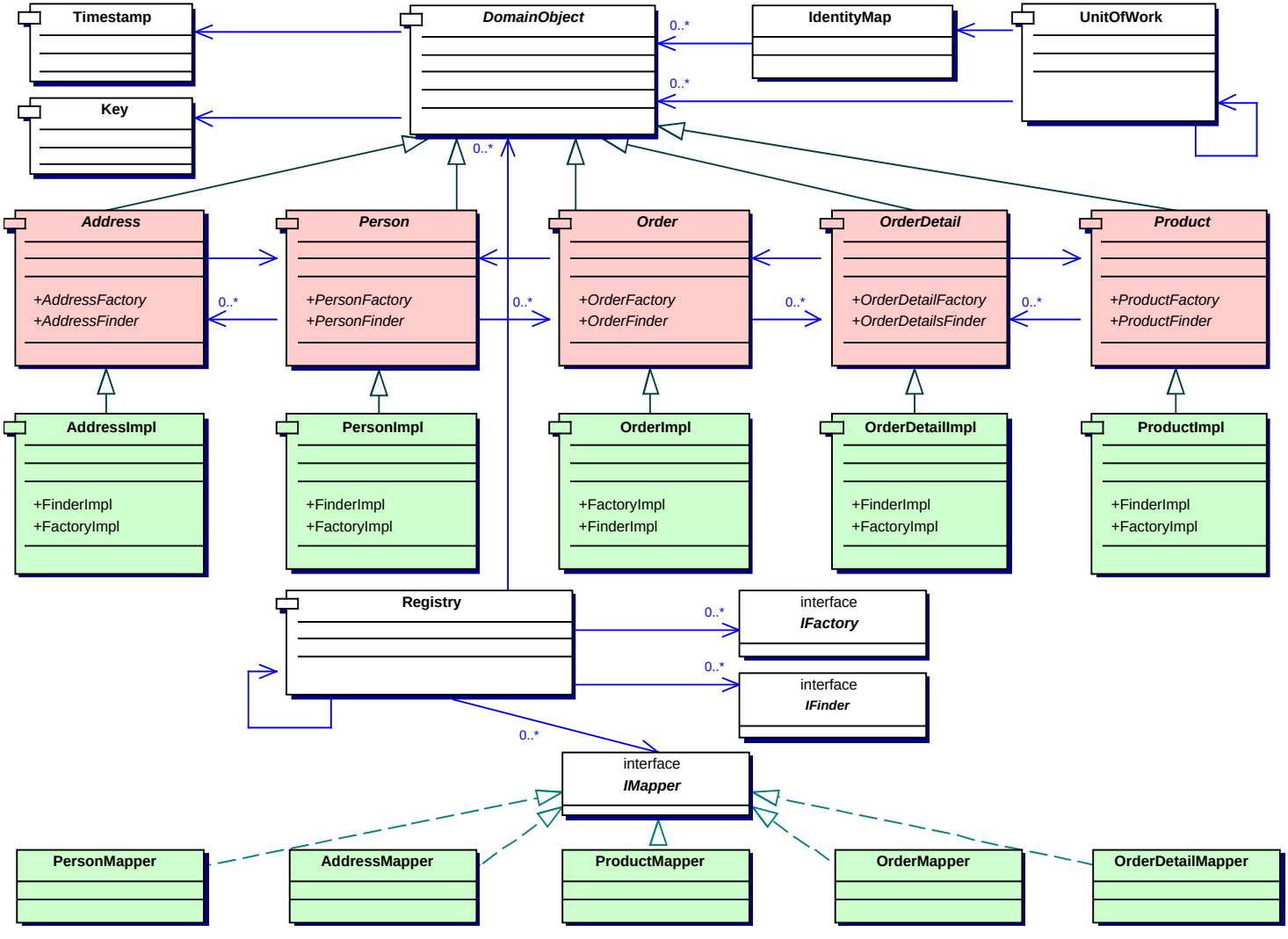
4.5 HsrOrderApp_S4

Das Design der HsrOrderApp_S4 wird vor allem durch die existierende Datenbank vorgegeben. Das Modell der HsrOrderApp_S3 wird wo möglich übernommen. Eine der markantesten Änderungen ist, dass die Domain Objekte in abstrakte Klassen und generierte Implementationen davon aufgeteilt werden.

Diese sind im konzeptionellen Modell auf der nächsten Seite rot (abstrakt) und grün (generiert).

Da sowieso alle Domain Objekte von der Oberklasse `DomainObject` abgeleitet werden müssen, haben wir entschieden den Primary Key (Klasse `Key`) und das Versions Feld (Klasse `Timestamp`) in die DomainObject Klasse auszulagern anstatt in die Implementationen zu generieren. Dies ist auch eine Erleichterung für den Programmierer, da er sie nicht in den abstrakten Klassen aufführen muss.

Das konzeptionelle Modell auf der nächsten Seite zeigt bei den Domain Objekten die internen Factory und Finder Klassen. Diese sind in den Abschnitten 4.3 und 4.4 beschrieben.


Timestamp
DomainObject
IdentityMap
UnitOfWork
Key
0..*
0..*
0..*
Address
Person
Order
OrderDetail
Product
+AddressFactory
+AddressFinder
+PersonFactory
+PersonFinder
+OrderFactory
+OrderFinder
+OrderDetailFactory
+OrderDetailsFinder
+ProductFactory
+ProductFinder
0..*
0..*
0..*
0..*
AddressImpl
PersonImpl
OrderImpl
OrderDetailImpl
ProductImpl
+FinderImpl
+FactoryImpl
+FinderImpl
+FactoryImpl
+FactoryImpl
+FinderImpl
+FinderImpl
+FactoryImpl
+FinderImpl
+FactoryImpl
Registry
interface
IFactory
interface
IFinder
0..*
0..*
0..*
interface
IMapper
PersonMapper
AddressMapper
ProductMapper
OrderMapper
OrderDetailMapper

Chapter 5

Enumeration Mapping

While the mapping of common data types is mostly clear we have to take a special look to enumerations. Enumeration is a type where numeric constants are represented by alphanumeric keywords or even descriptions. For developers the enum data type is a way to make the source code more readable. But most databases do not provide this type.

5.1 Enum data type

The enum data type has values given at compile time. If you want to change the values it is needed to recompile the code. Independent from how the mapping to the database is done there will ever be the problem of assuring the database consistency.

An object state like shown in Listing 5.1 is a place where enumeration makes really sense, because it makes the code much more readable.

If you don't want to run into problem you should use enumerators only for internal purposes never mapped to the database. Also the user of a program should never see any values of an enum.

5.2 Current mapping

The existing HsrOrderApp uses the enum type for a person role. In Figure 5.1 you can see the PersonRole table mapping PersonIDs to a RoleName. The

Listing 5.1: Object state with enumeration

```
public class DomainObject
{
  public enum ObjectState {Unchanged, Modified, Added, Deleted, Detached}

  private ObjectState m_state = ObjectState.Detached;

  internal ObjectState State
  {
    get { return m_state; }
    set { m_state = value; }
  }
}
```

value of the enum is written to the database as a string for each assignment.

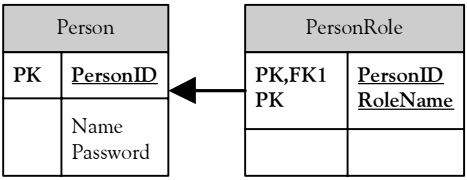


Figure 5.1: existing database

This is a bad database design because the strings are saved redundant. But not only that. You will also run into big troubles if you want to add, change or delete values of the enumerator. Every change must be done in the code and on the database to assure the consistency between enum values and database entries.

5.3 Normalized design

In a fully normalized database each string should be only saved once referenced by a unique id like you can see in Figure 5.2. There every role description exists exactly once. The table PersonRole now only maps RoleIDs to PersonIDs.

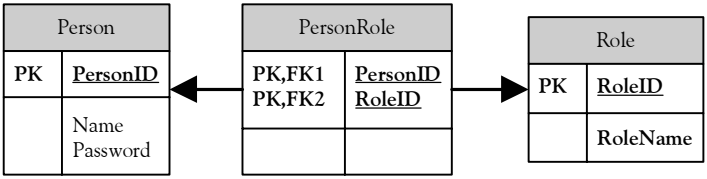


Figure 5.2: normalized database

5.4 Lookup tables

An enum is not dedicated to present a description to the user. Every text representing a database entry has to come out of the database. So you can avoid consistency problem between code and database.

Descriptions for numeric ids are always references to so called lookup table. You can add all descriptions to one table or use one tables for each relation. With a little effort it is even possible to save the values in different languages.

To change, add and remove the values you don't have to recompile the code but only change the entries in the database. Another advantage of a lookup table is the flexibility of such a table.

In Figure 5.3 you can see two tables which has its descriptions in the same table. The table Car has the names of colors and the table PersonRole its role names in the EnumLookup table. This descriptions can be added for different languages. The names of languages are defined in the Language table.

If descriptions of different tables are saved in one lookup table, you will need to know which descriptions belong to which table. There are two possibilities. You could define ranges of the ID belonging to a table. But the better solution is to add a description type to the lookup table as you can see in Figure 5.3. That is one of the rare cases where you could use an enum in the database if provided, because this values will only be used in the code. The user will never see it.

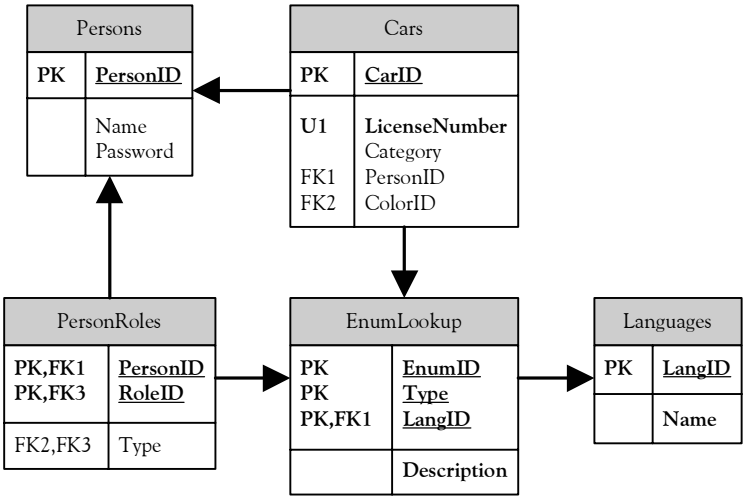


Figure 5.3: lookup table

5.5 Loading lookup table data

There are several strategies to load data from lookup tables. Due to the fact this data often are nearly static, it could be a good deal to read the data once and hold in memory.

Another possibility is to get the values within one request. In Listing 5.2 you see an example of joining the tables. This statement can be used directly or by creating a view on the database.

Listing 5.2: join tables

```
SELECT c.LicenseNumber, p.Name, e.Description AS Color
FROM Cars AS c
  LEFT OUTER JOIN Persons AS p
    ON c.PersonID = p.PersonID
  LEFT OUTER JOIN EnumLookup AS e
    ON c.ColorID = e.EnumID
      AND EnumLookup.Type = 'cars'
      AND EnumLookup.LangID = 1
WHERE c.Category = 1
```

Kapitel 6

Makefiles

Es ist nicht möglich, eine mit *STORM* geschriebene Applikation mit Visual Studio in einem Schritt zu erstellen. Das Problem liegt beim CustomTool von CodeSmith. Zuerst wird aus den abstrakten Klassen eine DLL gebildet. Diese wird von CodeSmith verwendet um den Code zu generieren. Danach sollte die DLL mit dem neu dazugekommenen Code neu erzeugt werden. Dies ist jedoch erst nach einem Neustart von Visual Studio möglich, da die DLL durch das CodeSmith CustomTool blockiert wird.

6.1 Lösungsansatz

Mit Makefiles hat man die Möglichkeit die Applikation oder Libraries in einem Schritt ohne das Aufstarten einer IDE zu erstellen. Änderungen in benötigten Dateien und Abhängigkeiten werden automatisch erkannt.

Wir haben zwei unabhängige Makefiles erstellt. Das eine übernimmt das Bilden der *STORM* DLL. Damit ist es ebenfalls möglich die DLL zu signieren und im GAC zu registrieren.

Mit dem zweiten Makefile ist es möglich ein Assembly aus den abstrakten Klassen zu bilden, daraus den benötigten Code zu generieren und dann die ganze HsrOrderApp Applikation zu kompilieren. Das ganze geschieht mit einem einzigen Befehl.

6.2 Storm DLL

Das Makefile für die *STORM* Library kann wie folgt ausgeführt werden:

```
nmake [clean] [keys] [build] [install] [uninstall]
```

clean: Löscht alle generierten Dateien aus dem Verzeichnis.
keys: Generiert ein Keypair.
build: Kompiliert und signiert den Code.
install: Registriert das Assembly im GAC.
uninstall: Entfernt das Assembly aus dem GAC.

In der Abbildung 6.1 sind die Abhängigkeiten der verschiedenen Schritte aufgezeigt. Wird zum Beispiel der Befehl `install` ausgeführt, wird auch überprüft ob das Keypair vorliegt und die Storm.dll mit den aktuellsten Source Dateien kompiliert wurde.

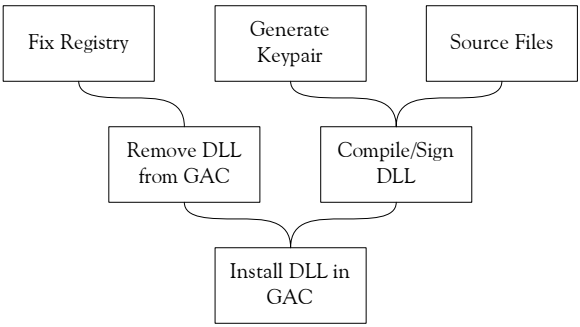


Abbildung 6.1: Storm Makefile

Auf einigen Windows Installationen gibt es einen fehlerhaften Registry Eintrag, der wahrscheinlich von einem Windows-Update stammt. Dieser verhindert, dass Assemblies aus dem GAC entfernt werden können. Das Makefile löscht diesen fehlerhaften Eintrag, bevor es versucht die DLL aus dem GAC zu entfernen. Der Eintrag ist der Folgende:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Installer\Assemblies\Global]
@="..."
```

6.3 HsrOrderApp

Das Makefile für die HsrOrderApp kann wie folgt ausgeführt werden:

```
nmake [clean] [impl] [mapper] [testapp] [run]
```

clean: Löscht alle generierten Dateien aus dem Verzeichnis.
impl: Generiert mit Hilfe von CodeSmith die Implementations-Klassen der Business-Objekte.
mapper: Generiert mit Hilfe von CodeSmith die Mapper-Klassen.
testapp: Kompiliert die Applikation.
run: Führt die Applikation aus.

Die Abbildung 6.2 zeigt die Abhängigkeiten beim bilden der HsrOrderApp. Die Generierung von Domain Object Implementierungen und Mapper Klassen wurde aufgeteilt, damit bei einer Änderung der Templates nur die betroffenen Dateien neu generiert werden müssen. In der Abbildung wurde auf diese Aufteilung verzichtet.

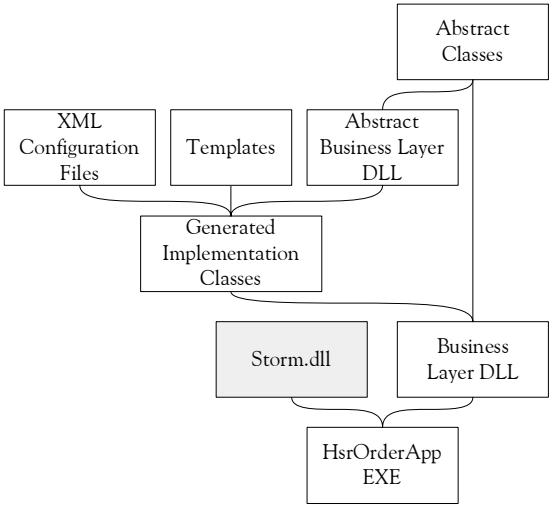


Abbildung 6.2: HsrOrderApp Makefile

Die Datei Storm.dll wird vorausgesetzt und wird nicht automatisch erstellt.

6.4 Erkenntnisse

Den Build Prozess mit Makefiles zu automatisieren wäre eine schöne Lösung, den Code auch ohne Visual Studio zu kompilieren. Ausserdem können andere Aufgaben wie Code Generierung, Keypair erstellen, Code signieren und DLL im GAC registrieren im Build Prozess eingebunden werden.

Hier hat uns jedoch CodeSmith einen Strich durch die Rechnung gemacht. Denn aus unerklärlichen Gründen, wird die DLL im GAC, während der Code Generierung, nicht gefunden. Wir sind daher gezwungen unsere DLL ins Programm Verzeichnis von CodeSmith zu kopieren, was alles andere als schön ist.

Teil III

Manuals

Der folgende Teil enthält Kapitel in Deutsch und Englisch. Wir haben entschieden, nur den technischen Bericht in Englisch zu verfassen, hatten zu diesem Zeitpunkt jedoch schon einzelne Kapitel geschrieben.

Chapter 7

CodeSmith

7.1 Template Syntax

CodeSmith [5] is a freeware template-based code generator which is able to generate code for any ASCII-based language. The syntax for creating CodeSmith templates is close to the ASP.NET syntax.

A template source file consists of template code and the static part of the desired source code output. In Listing 7.1 you can see an example. Everything encapsulated in `<% %>` tags will be replaced by CodeSmith, the rest be printed as is.

There are varieties of this tags. `<%@` is the beginning of a compiler directive. On the line 1 of Listing 7.1 you can see the `CodeTemplate` directive. This is the first and only mandatory code block in a template. It needs a `Language` attribute specifying the template language and a `TargetLanguage` attribute defining the language of generated code output. A `Description` attribute can be given optionally.

The `Property` directive you see at line 3 declares a property with a `Name` and a `Type` as mandatory attributes. This is exactly the same as a .NET property. The attribute `Category` defines the Section where the property is shown in the GUI. The given category on line 3 of Listing 7.1 can be found in the section title of the `ClassName` property in Figure 7.1. If no category is specified the property will be shown in the misc section.

A code block beginning with `<%=` contains only one property like at lines

Listing 7.1: Simple Example

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#"
                 Description="This is a sample template." %>
<%@ Property Name="ClassName" Type="System.String" Category="Design"
             Default="MyClass1" Description="Name of class to create" %>

public class <%= ClassName %>() {

    <%= ClassName %>() {
    }
}
```

6 and 8 in Listing 7.1. It is a short form to print that property. You can see the result of the template code in Listing 7.1 in Figure 7.1.

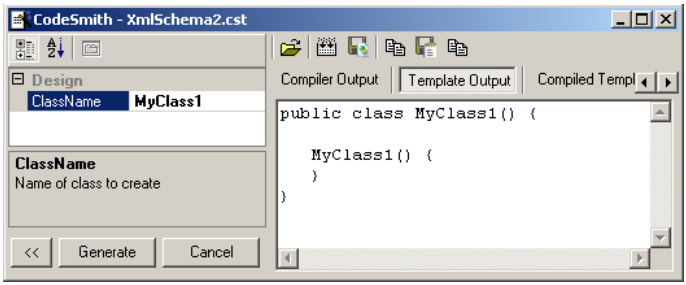


Figure 7.1: CodeSmith GUI

7.1.1 Functions

Template code can contain any logic like a loop for iterating, an `if` clause or anything else. If code is too complex to manage in a template there is the possibility to write functions. Usually they are placed at the end of a template in a `<script>` section as shown in Listing 7.2.

7.1.2 Derive from "code behind"

For more complex templates it could be useful to split the code into two files. The template file will only consists of static text and as few logic as possible.

Listing 7.2: <script> section

```
<script runat="template">
public string getEmail(string forename, string name)
{
  string email = forename + "." + name + "@hsr.ch";
  return email;
}
</script>
```

Only some simple loops or decisions. The main logic can be packed into a class placed in an external file. Such code is called "code behind" because you don't see it while working on the template.

This class has to be derived from the `CodeTemplate` class in the namespace `CodeSmith.Engine` as shown in Listing 7.3 at line 1 and 5. The code template has to inherit from this class. How to reference the external file in the template and inherit from the class in it is shown in Listing 7.4.

Listing 7.3: Code behind (sample1.cst.cs)

```
using CodeSmith.Engine;

namespace SampleNamespace
{
  public class Sample1 : CodeTemplate
  {
    private string className;

    public Sample1() { }

    public string ClassName {
      get{ return value; }
      set{ className = value; }
    }
  }
}
```

Listing 7.4: Include a base class (sample1.cst)

```
<%@ CodeTemplate Src="sample1.cst.cs" Inherits="SampleNamespace.Sample1"
                 Language="C#" TargetLanguage="C#" %>

public class <%= ClassName %>() {

   <%= ClassName %>() {
   }
}
```

Listing 7.5: Use an Assembly

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#" %>
<%@ Property Name="FileObject" Type="SampleNamespace.Sample2" %>
<%@ Assembly Name="ASample" %>

public SampleClass {
  public static void main() {
    Console.WriteLine("Selected file: {0}", <%= FileObject.File %>);
  }
}
```

Properties in a class you derive from must implement `set` and `get` methods because CodeSmith treats such a property like those declared with directives. CodeSmith has to read them to show the values in the GUI. And it has to manipulate them if you change a value over the GUI or generate code with an external configuration as described later.

All the properties and methods in such a class can be accessed directly from inline code as you see in Listing 7.4 at line 4 and 6.

7.1.3 Logic in an assembly

Another possibility to separating the logic from template code is to use an assembly. Listing 7.5 shows the usage of an assembly. There are two directives needed. The directive at line 3 specifies an assembly without the .dll extension. It needs to be in either the same directory as the template or in the same directory as the CodeSmith executable.

Now you can make instances of each class in that assembly. The line 2 in Listing 7.5 shows how to make an instance of the class `Sample2` in the namespace `SampleNamespace`. The property `FileObject` represents the instance. You can access all properties and methods of this object like shown in line 7 of Listing 7.5

The property defined in the template in that case is an object. This object can have several properties. That is why it is needed to implement an `Editor`, which enables you to set the properties of that object with the CodeSmith GUI. Such an `Editor` is a .NET technologie to edit the properties of an object which is bound to a input field in a GUI. If you bind your own object to an input field you have to write your own `Editor`.

For this the assembly must provide a form for a modal dialog or a drop-

down box which is designed to set all the properties. To show the settings in the GUI, CodeSmith calls the `ToString()` method. To have any effect on the text showed in the GUI you can overwrite the `ToString()` method.

Listings 7.6 and 7.7 use a default `OpenFileDialog` to demonstrate this functionality. Line 6 and 7 of Listing 7.6 is defined which `Editor` to use. Listing 7.7 shows the implementation of such an `Editor`.

Listing 7.6: Assembly source

```
using System;
using System.ComponentModel;

namespace SampleNamespace
{
  [Editor(typeof(SampleNamespace.Sample2Editor),
          typeof(System.Drawing.Design.UITypeEditor))]
  public class Sample2
  {
    private string m_file = "";

    public Sample2()
    {
    }

    public string File
    {
      get { return m_file; }
      set { m_file = value; }
    }

    public override string ToString() { return m_file; }
  }
}
```

Listing 7.7: Editor

```
using System;
using System.ComponentModel;
using System.Windows.Forms;
using System.Drawing.Design;

namespace SampleNamespace
{
  public class Sample2Editor : UITypeEditor
  {
    public Sample2Editor() : base() {}

    public override object EditValue(ITypeDescriptorContext context,
                                     IServiceProvider provider,
                                     object obj)
    {
      if (provider != null)
      {
```

```
18          OpenFileDialog fileDialog = new OpenFileDialog();

20          if(fileDialog.ShowDialog() == DialogResult.OK)
            {
22            obj = new Sample2();
              ((Sample2)obj).File = fileDialog.FileName;
24          }
          }
26        return obj;
        }
28
        public override
30      UITypeEditorEditStyle GetEditStyle(ITypeDescriptorContext context)
        {
32        return UITypeEditorEditStyle.Modal;
        }
34    }
  }
```

If you click into the input field in the CodeSmith GUI the method `EditValue()` will be called. This opens the `FileOpenDialog` and sets the property `File` after closing the dialog.

The method `GetEditStyle()` tells the CodeSmith GUI what kind of input box to provide. Either a drop-down box or a modal dialog are possible. For the `FileOpenDialog` you have to set `Modal` (see Listing 7.7,line 32). This results in a button with three dots like you can see next to the input box of `FileObject` in Figure 7.2.

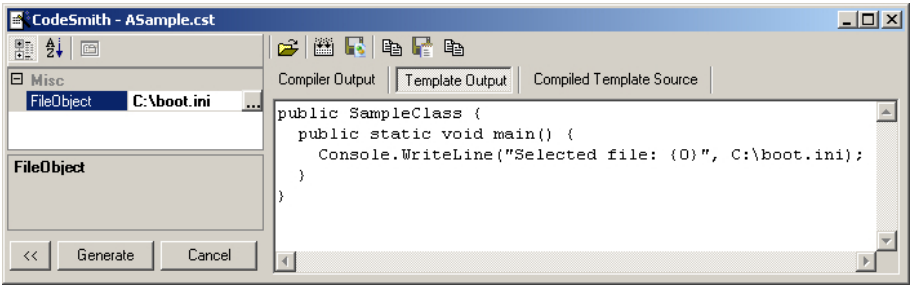


Figure 7.2: Input box for an assembly

Listing 7.8: xml configuration

```
<?xml version="1.0" encoding="utf-8" ?>
<codeSmith>
  <namespace>MyNamespace</namespace>
  <imports>
    <import namespace="System" />
    <import namespace="System.Collections" />
  </imports>
  <propertySets>
    <propertySet>
      <property name="FileName">BusinessObj1.dll</property>
    </propertySet>
    <propertySet>
      <property name="FileName">BusinessObj2.dll</property>
    </propertySet>
  </propertySets>
  <template path="BusinessObjects.cst" />
</codeSmith>
```

7.2 Code generation

If you want to generate code for several values it is a drawback that you have to set all the properties by hand. That's why you can can use an external file as input for the generation process. This procedure is described in the following section.

7.2.1 XML configuration file

There are two ways to generate code without the CodeSmith GUI. You can choose between a command line tool and the possibility to integrate the generation into Microsoft Visual Studio .NET. Both take the settings from an xml file like in Listing 7.8.

The specified `namespace` at line 3 is optional and encloses the whole generated code from all defined `propertySet`s. The `imports` given at lines 5 and 6 are included at the top of the generated code with the `using` keyword. For each `propertySet` defined in the `propertySets` element CodeSmith generates code with the properties specified in it.

The command line to generate code is:

```
CodeSmithConsole.exe sample1.xml output.cs
```

The first filename specifies the xml configuration file. The second filename is optional. If specified, the generated code is written to that file, otherwise

the output is shown on stdout.

7.2.2 Visual Studio .NET integration

To run the generation of code from within the Visual Studio you have to add the xml configuration file to the Visual Studio project. Then you have to set the property `Custom Tool` to `CodeSmithGenerator` as shown in Figure 7.3 on the right.

The `Build Action` field has to be set to `Content`. This causes Visual Studio to execute the generation every time the content of the xml file has changed. The generation of the code can also be initiated by right clicking the xml file and select `Run Custom Tool`.

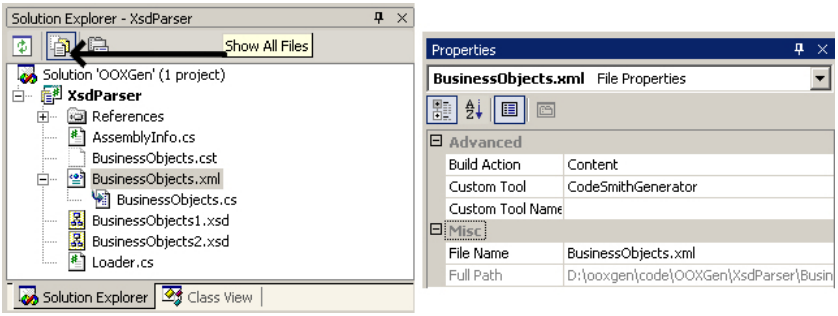


Figure 7.3: Visual Studio .NET integration

If the option *Show All Files* (see Figure 7.3) is turned on, the generated file is displayed as a child of the xml file.

Kapitel 8

STORM

8.1 Einleitung

Dieses Kapitel gibt eine Anleitung dazu, wie eine Applikation mit *STORM* erstellt werden kann. Der Grund dafür ist, dass die Verwendung der Attribute nicht ganz selbsterklärend ist. Die Attribute selbst wurden im technischen Bericht bereits beschrieben. Das wird hier nicht wiederholt.

8.2 Vorbereitung

8.2.1 CodeSmith

Zuerst muss CodeSmith auf dem Rechner installiert werden. Das Tool kann von http://www.ericjsmith.net/codesmith/download.aspx bezogen werden. Die Installation selbst sollte kein Problem darstellen.

8.2.2 Projekt erstellen

Als nächster Schritt wird ein neues Projekt oder eine neue Solution vom gewünschten Typ im Visual Studio .NET erstellt. Vorteilhaft ist es, wenn im neuen Projekt ein Ordner für die Domain Objekte angelegt wird. Dies ist zwar nicht nötig, aber dient der besseren Übersicht. Ebenso ist es nützlich,

gleich einen Ordner für die Domain Objekt Implementationen und für die Mapper Implementation zu machen.

8.2.3 Library und Templates

STORM beinhaltet zwei Templates, die für die Code Generierung benötigt werden. Dies sind `ImplGen.cst` und `MapperGen.cst`. Diese können grundsätzlich überall auf der lokalen Festplatte liegen. Es ist aber wiederum ratsam, einen eigenen Ordner im aktuellen Projekt zu erstellen und die beiden Dateien in diesen Ordner zu kopieren.

Des weiteren muss die STORM-Library (`Storm.dll`) in das von CodeSmith erstellte Verzeichnis kopiert werden. Dies ist leider nötig, da CodeSmith bei der Code Generierung auch dann die Dll nicht findet, wenn sie im GAC registriert wurde. Aktuell ist dieses CodeSmith Verzeichnis:

```
C:\Program Files\CodeSmith\v2.2\
```

Dies kann aber je nach lokalen Einstellungen und CodeSmith Version variieren.

Um eine Applikation mit *STORM* entwickeln zu können, muss die Library zusätzlich vom aktuellen Projekt referenziert werden. Dazu wird die Datei (`Storm.dll`) im Visual Studio .NET unter dem Punkt `References` → `Add References...` → `Browse` angegeben.

Figur 8.1 ist ein Beispiel wie eine Solution nach diesen Schritten aussehen sollte.

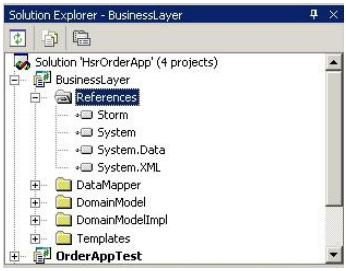


Abbildung 8.1: Solution Explorer nach der Vorbereitung

8.3 Abstrakte Klassen

Wenn alle Vorbereitungen abgeschlossen sind, kann man beginnen, die abstrakten Klassen zu schreiben. Natürlich muss vorher das Design der Applikation feststehen. Für jede Tabelle in der Datenbank muss eine Abstrakte Klasse geschrieben werden. Diese Klasse wird nachher mit den von *STORM* definierten Attributen versehen.

8.3.1 Übersicht

Hier eine Übersicht der behandelten Themen in diesem Kapitel:

Tabelle 8.1: Übersicht der Themen in diesem Abschnitt

8.3.2 Namespaces

Um mit den von *STORM* angebotenen Klassen arbeiten zu können, müssen als Erstes zwei Namespaces in der abstrakten Klasse eingebunden werden:

Listing 8.1: Namespaces einbinden

```
using Storm.Attributes;
using Storm.Lib;
```

8.3.3 Klassen-Attribute

Es gibt verschiedene Typen von Attributen. Einige sind für die Attributierung der Klassen vorgesehen, andere für Methoden, etc. Hier beginnen wir sinnvollerweise mit der Attributierung einer abstrakten Hauptklasse. Die Bezeichnung Hauptklasse verwenden wir deshalb, weil es Klassen innerhalb dieser Klasse geben kann. Die folgende Liste beschreibt alle Attribute die für eine Hauptklasse verwendet werden. Es müssen alle Attribute angegeben werden.

- Table(string tableName, bool keyIsSurrogate)
  Wird benötigt um das Mapping zwischen einer Klasse und einer Tabelle zu machen. Zu jeder Klasse gehört eine Tabelle in der Datenbank.

  tableName Der Name der Tabelle in der Datenbank, auf die diese Klasse gemappt werden soll.

  keyIsSurrogate Definiert ob ein Key surrogate, also künstlich ist oder nicht. Falls `true` angegeben wird, darf in der Klasse nur ein Primary Key definiert werden. Ansonsten dürfen mehrere Primary Keys definiert werden.

- VersionField(string fieldName)
  Das Versionen Feld wird benötigt um die Aktualität der gespeicherten Objekte zu überprüfen.

  fieldName Der Name des Feldes in der Datenbank. Erwartet wird hier ein Feld vom Typ `timestamp`. Dies entspricht einer binären Nummer, die garantiert über die ganze Datenbank eindeutig ist.

- GenerateCode()
  Dieses Attribut wurde zur Sicherheit eingeführt. Es muss angegeben werden, falls Code für diese Klasse generiert werden sollte. Falls dieses Attribut fehlt, wird die Klasse von *STORM* ignoriert.

Listing 8.2 zeigt ein Beispiel für eine Deklaration einer Hauptklasse. Zu Beachten ist dabei auch, dass die Hauptklasse von `DomainObject` abgeleitet sein muss.

Listing 8.2: Definition einer Hauptklasse

```
[Table("Persons", true),
VersionField("chTimestamp"),
GenerateCode]
public abstract class Person : DomainObject
{
  ...
}
```

8.3.4 interne Klassen

Wie oben angedeutet, ist es möglich bzw. nötig, Klassen innerhalb der Hauptklasse zu deklarieren. Dies ist möglich für zwei Fälle:

- Eine Factory Klasse kann deklariert werden. Diese wird dazu benutzt, eigene Konstruktoren zu erstellen. Falls diese Klasse nicht definiert wurde, kann nur der von *STORM* definierte Konstruktor aufgerufen werden.

- Eine Finder Klasse kann deklariert werden. Diese Klasse dient dazu, eigene Suchmethoden zu definieren. Von *STORM* sind die drei Funktionen `find`, `findAll` und `findById` vordefiniert. Diese können auch aufgerufen werden, wenn die Finder Klasse nicht deklariert wurde. Der `find` Methode kann ein QueryObject mitgegeben werden. Damit ist dieser Aufruf an sich schon sehr flexibel. Trotzdem ist es aber in manchen Fällen angenehmer, eigene Methoden aufrufen zu können.

Factory Klassen müssen das Attribute Factory haben. Innerhalb dieser Klasse werden selbst definierte Konstruktoren definiert. Zu Beachten gilt, dass die Klasse als abstrakt deklariert werden muss. Ein Beispiel für eine Factory Klasse ist das folgende Listing:

Listing 8.3: Definition Factory Klasse

```
public abstract class Person : DomainObject
{
  [Factory]
  public abstract class PersonFactory
  {
    public abstract Person createPerson(
      [ParameterDef("Name")] string name,
      [ParameterDef("Password")] string password);
  }
}
```

Es können beliebig viele solcher Konstruktoren deklariert werden. Damit klar ist, worauf sich ein Parameter einer solchen Methode bezieht, muss pro Parameter ein ParameterDef Attribute angegeben werden. Die Bedeutung der benutzten Attribute ist Folgende:

- Factory()
  Das Attribute gibt an, dass die folgende Klasse eine interne Klasse der Hauptklasse ist und dass eine Factory Klasse folgt. In einer Factory Klasse können selbst definierte Konstruktoren deklariert werden.

- ParameterDef(string propertyName)
  Beschreibt einen Parameter einer Methode. Jedem Parameter muss ein solches Attribut vorangehen.

propertyName Gibt an, auf welches Property sich dieser Parameter bezieht. Das Property muss in der Hauptklasse existieren. Properties sind in Abschnitt 8.3.5 Beschrieben.

Finder Klassen können sehr ähnlich deklariert werden. Sie müssen ebenfalls abstrakt sein. Ein Beispiel ist nachstehend angegeben:

Listing 8.4: Definition Factory Klasse

```
public abstract class Person : DomainObject
{
  [Finder]
  public abstract class PersonFinder
  {
    public abstract IList findByNameAndPassword(
      [ParameterDef("Name")] string name,
      [ParameterDef("Password")] string password);
  }
}
```

Im Gegensatz zu den Factory Methoden, muss bei einer Finder Klasse das Finder Attribut angegeben werden. Dieses Attribut gibt an, dass die darauf folgenden Klasse selbst definierte Finder Methoden enthält. Das ParameterDef Attribute wurde bereits beschrieben und muss auch hier pro Parameter angegeben werden.

8.3.5 Properties

Für jede Kolonne in der Datenbank muss ein Property in der abstrakten Klasse deklariert werden. Falls ein Property nicht angegeben wird, ist auch die Kolonne in der Datenbank nicht benutzbar. Zusätzlich kann es zu Schwierigkeiten führen, falls das Attribut benötigt wird (z.B. für Relationen). Ein Beispiel einer Deklaration eines Property ist folgendes:

Listing 8.5: Definition eines Property

```
[Column("Name")]
public abstract string Name {get; set;}
```

Properties müssen abstrakt sein. Sie werden im generierten Code implementiert. Es kann mit `get;` bzw. `set;` angegeben werden, was von einem Property implementiert werden soll. Falls von einem anderen Attribut (z.B. ParameterDef Attribut) auf ein Property verwiesen wird, muss das Property eine `set` Methode haben. Column ist das einzige Attribut das zwingend ist. Die Bedeutung ist folgende:

- Column(string dbColumn)
  Dieses Attribut wird verwendet, um ein Property auf eine Kolonne in der Datenbank zu mappen.

  dbColumn Der Name der dazugehörenden Kolonne in der Datenbank.

8.3.6 Primary Key(s)

Ein Primary Key wird wie ein Property (8.3.5) deklariert. Zusätzlich zu einem gewöhnlichen Property kommt aber noch ein PrimaryKey Attribut hinzu. Falls im Table Attribut `isSurrogateKey` als `false` gesetzt wurde, können mehrere Properties das Attribut PrimaryKey haben. Ansonsten darf dieses Attribut nur einmal verwendet werden. Dieses Attribut hat keine Parameter. Ein Beispiel einer Deklaration eines Primary Keys ist folgendes:

Listing 8.6: Definition eines Property

```
[Column("PersonID"), PrimaryKey]
public abstract int PersonId {get;}
```

8.3.7 ToMany Relation

Eine ToMany Relation wird ebenfalls mit einem Property deklariert. Es ist aber kein Column Attribute nötig, da diese Relation auf eine andere Klasse verweist. Eine ToMany Relation ist eine one-to-many Relation. Für jede ToMany Relation muss es eine Adder Methode (8.3.9) geben.

Listing 8.7: Definition eines Property

```
[ToMany(typeof(Address), "Person")]
public abstract IList Addresses {get;}
```

Die Bedeutung des Attributes ist folgende:

- ToMany(Type relationTo, string relationName)
  Dieses Attribut zeigt an, dass es sich beim diesem Property um eine one-to-many Relation handelt.

  relationTo Hier muss der Typ der Klasse angegeben werden, auf die sich die Relation bezieht, also der to-many Teil der Relation.

relationName Dies gibt den Namen des Property in der Klasse an, auf die sich die Relation bezieht. In dieser Klasse muss ein Property mit diesem Name existieren. Zudem muss dieses Property ein ToOne Attribut besitzen.

8.3.8 ToOne Relation

Eine ToOne Relation wird entweder für eine back-Referenz einer ToMany Relation oder für eine one-to-one Relation benutzt. Es muss für *jede* ToMany Relation auch eine ToOne Relation geben. Das folgende Beispiel zeigt die Verwendung dieser beiden Attribute für eine Person mit Adressen:

Listing 8.8: Zusammenhang zwischen ToMany und ToOne Attribut

```
public abstract class Person : DomainObject
{
  [Column("PersonID"), PrimaryKey]
  public abstract int PersonId {get;}

  [ToMany(typeof(Address), "Person")]
  public abstract IList Addresses {get;}
}

public abstract class Address : DomainObject
{
  [Column("PersonID"), ToOne(typeof(Person), "PersonId")]
  public abstract Person Person {get; set;}
}
```

Zu Beachten gilt, dass das mit ToOne attributierte Property ein Column Attribute besitzen muss. Zusätzlich muss das Property eine `set` Methode haben.

Die Bedeutung des ToOne Attributes ist folgende:

- ToOne(Type relationTo, string relationName)
  Dieses Attribut zeigt an, dass es sich beim diesem Property entweder um eine one-to-one Relation oder um eine back-Referenz einer one-to-many Relation handelt. Es muss ein Column Attribut und eine `set` Methode deklariert werden.

  relationTo Hier muss der Typ der Klasse angegeben werden, auf die sich die Relation bezieht, also der one-to Teil der Relation.

  relationName Dies gibt den Namen des Property in der Klasse an, auf die sich die Relation bezieht. Diese Relation muss sich auf einen Primary Key beziehen (siehe Beispiel).

8.3.9 Adder Methoden

Bei ToMany Relationen darf beim Property keine `set` Methode deklariert werden, da der Rückgabetyp nicht mit dem zu übergebenden Typ übereinstimmt. Aus diesem Grund wurden Adder Methoden eingeführt. Das bedeutet, es sollte für jede ToMany Relation auch eine Adder Methode definiert werden, um tatsächlich Objekte der Relation hinzufügen zu können. Die Methode muss abstrakt sein. Ein Beispiel einer Deklaration einer solchen Adder Methode ist folgende:

Listing 8.9: Zusammenhang zwischen ToMany und ToOne Attribut

```
public abstract class Person : DomainObject
{
  [ToMany(typeof(Address), "Person")]
  public abstract IList Addresses {get;}

  [Adder("Addresses", "Person")]
  public abstract void addAddress(Address a);
}
```

Die Bedeutung des Attributes ist folgende:

- Adder(string localProperty, string targetProperty)
  Dieses Attribut wird in Ergänzung zu ToMany Relationen verwendet. Es dient dazu Methoden deklarieren zu können, die es erlauben, einer ToMany Relationen Objekte hinzuzufügen.

  localProperty Der Name des Property, der die dazugehörende ToMany Relation deklariert. Diese ist in derselben abstrakten Klasse.

  targetProperty Dies gibt den Namen des Property in der Klasse an, auf die sich die ToMany Relation bezieht. Dieser Name ist derselbe, der im ToMany Attribut angegeben wird.

8.3.10 Ein Beispiel

Da bis jetzt nur Ausschnitte aus der abstrakten Klasse gezeigt wurden, wird hier ein zusammenhängendes Beispiel einer abstrakten Klasse gegeben. Es ist das Beispiel einer Person:

Listing 8.10: Zusammenhang zwischen ToMany und ToOne Attribut

```
[Table("Persons", true),
  VersionField("chTimestamp"),
```

```
  GenerateCode]
public abstract class Person : DomainObject
{
  [Factory]
  public abstract class PersonFactory
  {
    public abstract Person createPerson(
      [ParameterDef("Name")] string name,
      [ParameterDef("Password")] string password);
  }

  [Finder]
  public abstract class PersonFinder
  {
    public abstract IList findByName([ParameterDef("Name")] string name);
    public abstract IList findByNameAndPassword(
      [ParameterDef("Name")] string name,
      [ParameterDef("Password")] string password);
  }

  [Column("PersonID"), PrimaryKey]
  public abstract int PersonId {get;}

  [Column("Name")]
  public abstract string Name {get; set;}

  [Column("Password")]
  public abstract string Password {get; set;}

  [ToMany(typeof(Address), "Person")]
  public abstract IList Addresses {get;}

  [ToMany(typeof(Order), "Person")]
  public abstract IList Orders {get;}

  [Adder("Addresses", "Person")]
  public abstract void addAddress(Address a);

  [Adder("Orders", "Person")]
  public abstract void addOrder(Order o);
}
```

8.3.11 Zusätzliche Methoden

In der abstrakten Klasse können natürlich auch Methoden implementiert werden. Diese werden nicht attributiert und damit von *STORM* ignoriert. Solche Methoden können aber dennoch wie gewohnt benutzt werden.

8.4 Generieren und Kompilieren

Nachdem alle abstrakten Klassen geschrieben wurden, kann das Projekt kompiliert werden. Falls es ohne Fehler kompiliert, können die Domain Objekt Implementationen und die Mapper Implementationen generiert werden. Da es für jede erstellte abstrakte Klasse jeweils zwei Implementations-Klassen generiert werden, muss es entsprechend viele XML Konfigurations-Dateien geben. Wie das aussehen könnte ist in Abbildung 8.2 gezeigt.

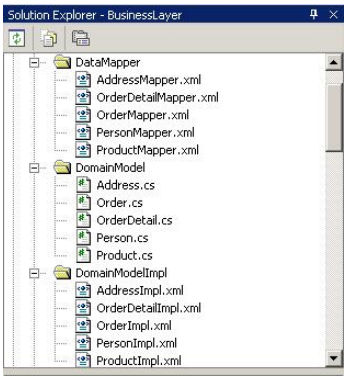


Abbildung 8.2: Solution Explorer mit den Konfigurations-Dateien

Eine Beschreibung um den Code zu generieren ist im CodeSmith Manual ab Abschnitt 7.2 nachzuschlagen.

Nachdem alle Klassen generiert wurden, muss das Visual Studio .NET geschlossen werden. Das ist, weil CodeSmith die dll blockiert und deshalb kann nicht mehr darauf zugegriffen werden. Nachdem das Projekt wieder geöffnet wurde, kann das gesamte Projekt, mit den erstellten Klassen, kompiliert werden.

8.5 Arbeiten mit dem Projekt

Es wurde bereits im technischen Bericht ein Beispiel gegeben, wie die Klassen des neu erstellten Projektes benutzt werden können. Es wird deshalb hier nicht wiederholt, zumal die Benutzung auch von den erstellten Abstrak-

ten Klassen abhängt. Wichtig ist aber immer, dass die generierten Klassen niemals direkt benutzt werden, sonder immer über die Factory bezogen werden.

Teil IV

Projektplanung

Kapitel 9

Risiken

Neben den üblichen Risiken, die in jedem Projekt bestehen (Krankheit, persönliche Probleme, etc.) bestehen folgende Risiken:

Tabelle 9.1: Risiken

Nr	$P[\%]$	**Beschreibung**	**Konsequenz**
1	30	Projektplan (Meilensteine) kann nicht eingehalten werden	Funktionalitäten der Applikation müssen gestrichen werden. Welche Funktionalitäten davon betroffen sind kann nicht allgemein formuliert werden. Dies würde je nach Zeitverzögerung entschieden werden. Wichtig dabei ist das frühe Reagieren.
2	30	Die gewählten Ansätze führen zu keinem erfolgreichen Ziel	Die Erkenntnisse müssen dokumentiert werden, die Arbeit wird aber trotzdem aus Zeitgründen mit den gewählten Ansätzen zu Ende geführt.

3	20	Technologiestudium beansprucht zu viel Zeit	Der Zeitplan muss angepasst werden, was Risiko 1 zur Folge hat.
4	10	CodeSmith erfüllt die Anforderungen nicht	Um dies zu verhindern, wir ein früher Meilenstein gesetzt. Dieser hat zum Ziel, CodeSmith zu testen und seine Tauglichkeit zu überprüfen. Falls festgestllt wird, dass CodeSmith die Anforderungen nicht erfüllt, muss ein alternatives Tool gefunden werden.

Kapitel 10

Projektplan

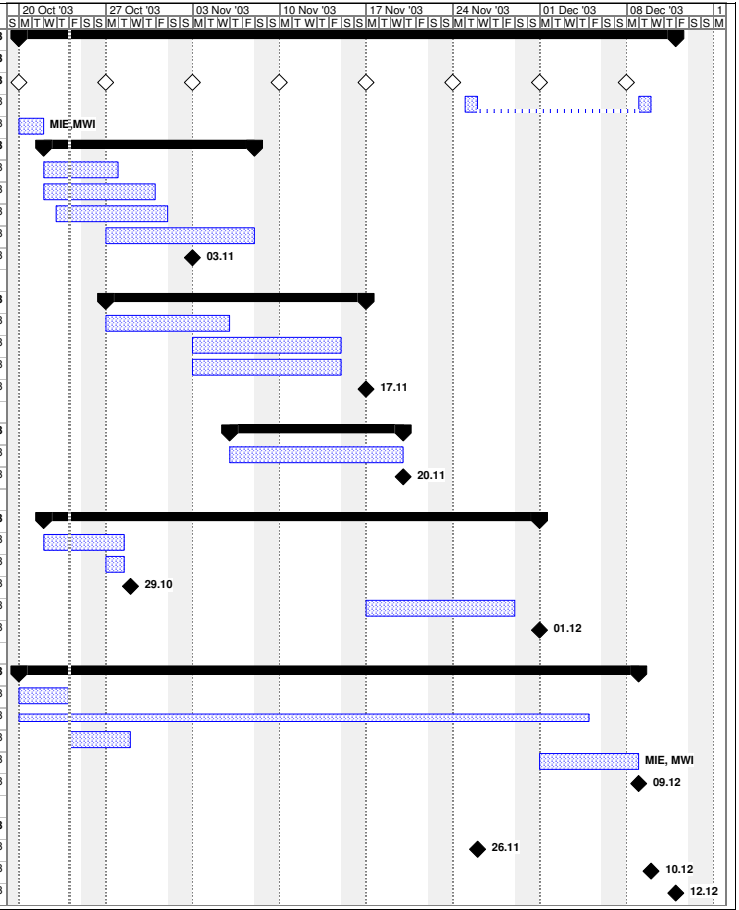
10.1 Versionen

Die Projektpläne sind auf den nächsten Seiten der Dokumentation beigelegt. Folgende Versionen sind verfügbar:

Tabelle 10.1: Versionen des Projektplans

Version	Erstellt am	Geändert
0.1	24. Oktober 2003	Neu erstellt
0.2	28. Oktober 2003	Im Teil Implementation den Task „CodeSmit Templates erstellen hinzugefügt“
0.3	03. November 2003	Task „Anwenungen Patterns“ gelöscht. Task „CodeSmith Templates erstellen“ früher angesetzt, um frühzeitig konkrete Resultate zu erzielen.
0.4	18. November 2003	Task „Anwendungen generische Ansätze“ aufgesplittet. Meilenstein 3 ist dadurch später. Task „Templates erstellen“ dem Task „HsrOrderApp_S4“ angepasst.

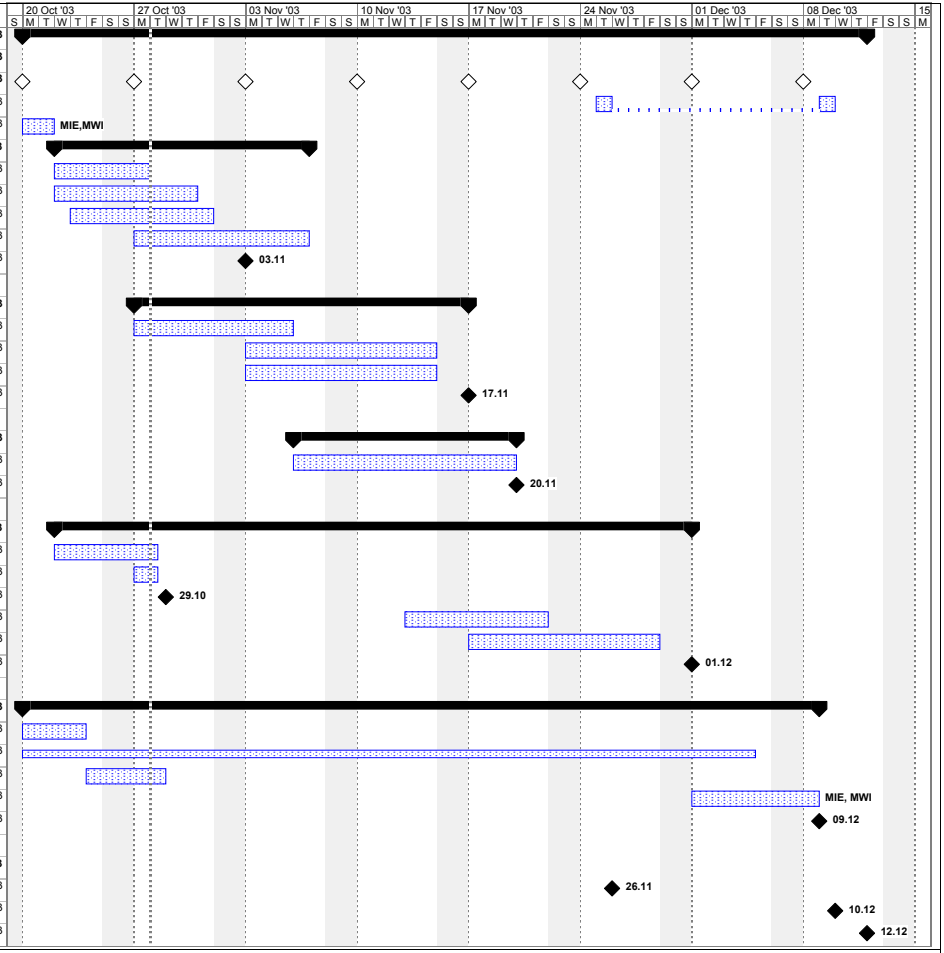
ID	Task Name	Duration	Start
1	**Tasks**	**39 days?**	**Mon 20.10.03**
2	**Allgemeines**	**37 days**	**Mon 20.10.03**
3	**Meetings**	**35 days**	**Mon 20.10.03**
12	Vorbereitung Präsentationen	2 days	Tue 25.11.03
13	IT Installation (HW, SW)	2 days	Mon 20.10.03
14	**Technologiestudium**	**13 days?**	**Wed 22.10.03**
15	Codesmith	4 days?	Wed 22.10.03
16	Patterns	7 days?	Wed 22.10.03
17	XML Schema	7 days?	Thu 23.10.03
18	.NET Technologien	10 days?	Mon 27.10.03
19	Meilenstein 2	0 days	Mon 03.11.03
20			
21	**Analyse**	**15 days?**	**Mon 27.10.03**
22	HSROrderApp_S3	8 days?	Mon 27.10.03
23	Anwendungen Patterns	10 days?	Mon 03.11.03
24	Anwenudngen generische / generative Ansätze	10 days?	Mon 03.11.03
25	Meilenstein 3	0 days	Mon 17.11.03
26			
27	**Design**	**10 days?**	**Thu 06.11.03**
28	HSROrderApp_S4	10 days?	Thu 06.11.03
29	Meilenstein 4	0 days	Thu 20.11.03
30			
31	**Implementation**	**28 days?**	**Wed 22.10.03**
32	CodeSmith Prototyp	4.5 days?	Wed 22.10.03
33	Einbindung .NET Studio	1.5 days?	Mon 27.10.03
34	Meilenstein 1	0 days	Wed 29.10.03
35	HSROrderApp_S4	10 days?	Mon 17.11.03
36	Meilenstein 5	0 days	Mon 01.12.03
37			
38	**Dokumentation**	**36 days?**	**Mon 20.10.03**
39	Project Plan	4 days	Mon 20.10.03
40	Allgemein (Vision, Glossar, etc.)	34 days	Mon 20.10.03
41	CodeSmith	3 days?	Fri 24.10.03
42	Abstract, Mgmt Summary, etc.	6 days	Mon 01.12.03
43	Meilenstein 6	0 days	Tue 09.12.03
44			
45	**Meilensteine Allgemein**	**12 days**	**Wed 26.11.03**
46	Zwischenpräsentation	0 days	Wed 26.11.03
47	Abgabe Diplomarbeit	0 days	Wed 10.12.03
48	Ausstellung und Vorführung	0 days	Fri 12.12.03



Project: ProjectPlan
Date: Fri 24.10.03

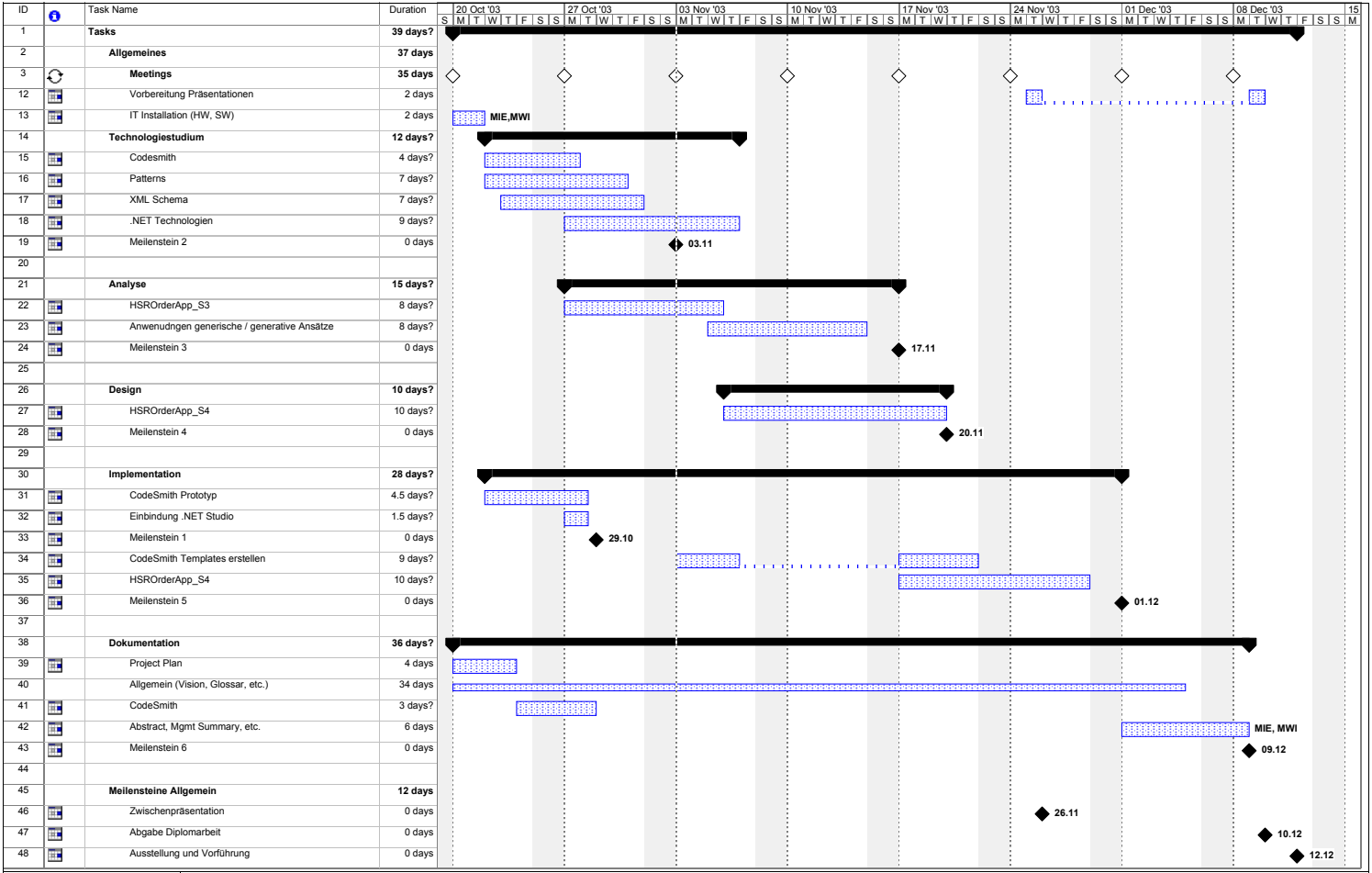
Task | Progress | Summary | External Tasks | Deadline
Split | Milestone | Project Summary | External Milestone

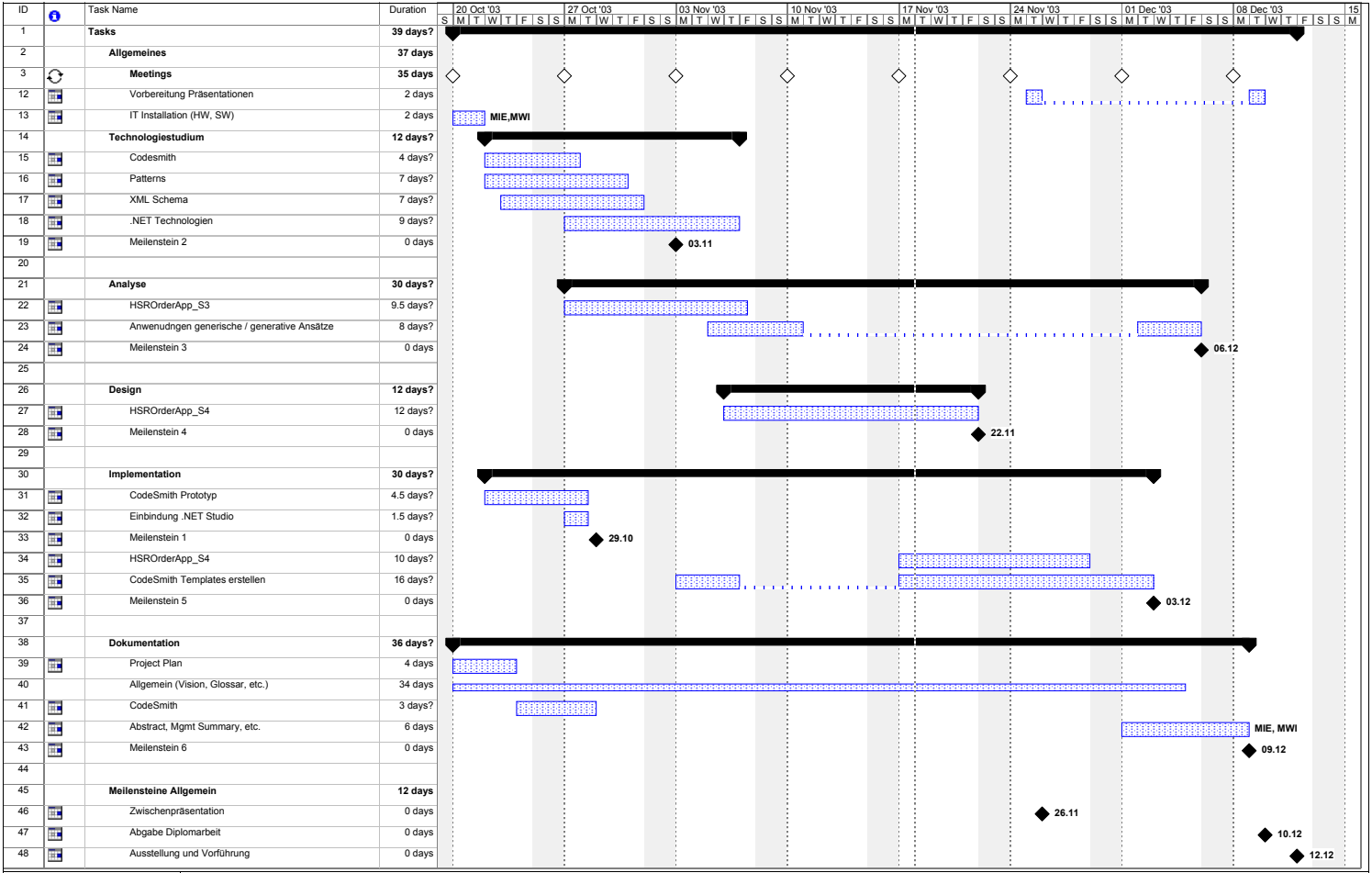
ID	Task Name	Duration	Start
1	**Tasks**	**39 days?**	**Mon 20.10.03**
2	**Allgemeines**	**37 days**	**Mon 20.10.03**
3	**Meetings**	**35 days**	**Mon 20.10.03**
12	Vorbereitung Präsentationen	2 days	Tue 25.11.03
13	IT Installation (HW, SW)	2 days	Mon 20.10.03
14	**Technologiestudium**	**12 days?**	**Wed 22.10.03**
15	Codesmith	4 days?	Wed 22.10.03
16	Patterns	7 days?	Wed 22.10.03
17	XML Schema	7 days?	Thu 23.10.03
18	.NET Technologien	9 days?	Mon 27.10.03
19	Meilenstein 2	0 days	Mon 03.11.03
20			
21	**Analyse**	**15 days?**	**Mon 27.10.03**
22	HSROrderApp_S3	8 days?	Mon 27.10.03
23	Anwendungen Patterns	10 days?	Mon 03.11.03
24	Anwenudngen generische / generative Ansätze	10 days?	Mon 03.11.03
25	Meilenstein 3	0 days	Mon 17.11.03
26			
27	**Design**	**10 days?**	**Thu 06.11.03**
28	HSROrderApp_S4	10 days?	Thu 06.11.03
29	Meilenstein 4	0 days	Thu 20.11.03
30			
31	**Implementation**	**28 days?**	**Wed 22.10.03**
32	CodeSmith Prototyp	4.5 days?	Wed 22.10.03
33	Einbindung .NET Studio	1.5 days?	Mon 27.10.03
34	Meilenstein 1	0 days	Wed 29.10.03
35	CodeSmith Templates erstellen	7 days?	Thu 13.11.03
36	HSROrderApp_S4	10 days?	Mon 17.11.03
37	Meilenstein 5	0 days	Mon 01.12.03
38			
39	**Dokumentation**	**36 days?**	**Mon 20.10.03**
40	Project Plan	4 days	Mon 20.10.03
41	Allgemein (Vision, Glossar, etc.)	34 days	Mon 20.10.03
42	CodeSmith	3 days?	Fri 24.10.03
43	Abstract, Mgmt Summary, etc.	6 days	Mon 01.12.03
44	Meilenstein 6	0 days	Tue 09.12.03
45			
46	**Meilensteine Allgemein**	**12 days**	**Wed 26.11.03**
47	Zwischenpräsentation	0 days	Wed 26.11.03
48	Abgabe Diplomarbeit	0 days	Wed 10.12.03
49	Ausstellung und Vorführung	0 days	Fri 12.12.03



Project: ProjectPlan_v0-2
Date: Tue 28.10.03

Task | Progress | Summary | External Tasks | Deadline
Split | Milestone | Project Summary | External Milestone


ID
Task Name
Duration
20 Oct '03
27 Oct '03
03 Nov '03
10 Nov '03
17 Nov '03
24 Nov '03
01 Dec '03
08 Dec '03
1 Tasks 39 days?
2 Allgemeines 37 days
3 Meetings 35 days
12 Vorbereitung Präsentationen 2 days
13 IT Installation (HW, SW) 2 days
MIE,MWI
14 Technologiestudium 12 days?
15 Codesmith 4 days?
16 Patterns 7 days?
17 XML Schema 7 days?
18 .NET Technologien 9 days?
19 Meilenstein 2 0 days
03.11
21 Analyse 15 days?
22 HSROrderApp_S3 8 days?
23 Anwenudngen generische / generative Ansätze 8 days?
24 Meilenstein 3 0 days
17.11
26 Design 10 days?
27 HSROrderApp_S4 10 days?
28 Meilenstein 4 0 days
20.11
30 Implementation 28 days?
31 CodeSmith Prototyp 4.5 days?
32 Einbindung .NET Studio 1.5 days?
33 Meilenstein 1 0 days
29.10
34 CodeSmith Templates erstellen 9 days?
35 HSROrderApp_S4 10 days?
36 Meilenstein 5 0 days
01.12
38 Dokumentation 36 days?
39 Project Plan 4 days
40 Allgemein (Vision, Glossar, etc.) 34 days
41 CodeSmith 3 days?
42 Abstract, Mgmt Summary, etc. 6 days
MIE, MWI
43 Meilenstein 6 0 days
09.12
45 Meilensteine Allgemein 12 days
46 Zwischenpräsentation 0 days
26.11
47 Abgabe Diplomarbeit 0 days
10.12
48 Ausstellung und Vorführung 0 days
12.12
Project: ProjectPlan
Date: Mon 03.11.03
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline
Page 1


ID
Task Name
Duration
20 Oct '03
27 Oct '03
03 Nov '03
10 Nov '03
17 Nov '03
24 Nov '03
01 Dec '03
08 Dec '03
1 Tasks 39 days?
2 Allgemeines 37 days
3 Meetings 35 days
12 Vorbereitung Präsentationen 2 days
13 IT Installation (HW, SW) 2 days
MIE,MWI
14 Technologiestudium 12 days?
15 Codesmith 4 days?
16 Patterns 7 days?
17 XML Schema 7 days?
18 .NET Technologien 9 days?
19 Meilenstein 2 0 days
03.11
21 Analyse 30 days?
22 HSROrderApp_S3 9.5 days?
23 Anwenudngen generische / generative Ansätze 8 days?
24 Meilenstein 3 0 days
06.12
26 Design 12 days?
27 HSROrderApp_S4 12 days?
28 Meilenstein 4 0 days
22.11
30 Implementation 30 days?
31 CodeSmith Prototyp 4.5 days?
32 Einbindung .NET Studio 1.5 days?
33 Meilenstein 1 0 days
29.10
34 HSROrderApp_S4 10 days?
35 CodeSmith Templates erstellen 16 days?
36 Meilenstein 5 0 days
03.12
38 Dokumentation 36 days?
39 Project Plan 4 days
40 Allgemein (Vision, Glossar, etc.) 34 days
41 CodeSmith 3 days?
42 Abstract, Mgmt Summary, etc. 6 days
MIE, MWI
43 Meilenstein 6 0 days
09.12
45 Meilensteine Allgemein 12 days
46 Zwischenpräsentation 0 days
26.11
47 Abgabe Diplomarbeit 0 days
10.12
48 Ausstellung und Vorführung 0 days
12.12
Project: ProjectPlan
Date: Tue 18.11.03
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline

10.2 Änderungsbeschreibung

10.2.1 Version 0.2

Für die CodeSmith Templates wurde sinnvollerweise ein eigener Task hinzugefügt.

10.2.2 Version 0.3

Eine Dokumentation von Anwendungen der benutzten Patterns wird nicht benötigt. Erstens kann im Rahmen dieser zeitlich kurzen Arbeit nicht genügend Erfahrung mit verschiedenen Applikationen gesammelt werden, damit eine solche Dokumentation Sinn machen würde, und zweitens gibt es bereits viele Bücher bzw. Berichte die dieses Thema behandeln.

Zudem werden schon früher als geplant erste CodeSmith Templates geschrieben, damit Risiken diesbezüglich minimiert werden können.

10.2.3 Version 0.4

Meilenstein 3 (siehe Beschreibung Meilenstein 3) wurde nach hinten verschoben. Der Grund dafür ist, dass es sich bei diesem Meilenstein um die Dokumentation von Erkenntnissen handelt, die wir erst am Schluss in vollem Umfange besitzen können.

Der Task „Templates erstellen“ wurde an die Implementation der HsrOrderApp_S4 angepasst. Es hat sich gezeigt, dass es sinnvoll ist, wenn zuerst die Applikation von Hand geschrieben wird und erst anschliessend die entsprechenden Templates dafür geschrieben werden. Das hat zu Konsequenz, dass die Templates erst nach der Implementation der Applikation fertig gestellt werden können.

Kapitel 11

Zeitplanung

11.1 Geplante Meilensteine

Es sind insgesamt 6 selbst definierte und 3 vorgegebene Meilensteine definiert. Die Meilensteine stimmen mit diesen im Projektplan überein. Im Folgenden werden diese Meilensteine und deren Ziele erläutert.

11.1.1 Selbst definierte Meilensteine

Meilenstein 1 Ein CodeSmith Prototyp wird erstellt. Dies beinhaltet die Einbindung eines Custom Tools in das Visual Studio .NET und die Generierung eines einfachen Codefiles aus der Definition einer XML Schema Datei. Zur Konfiguration soll eine XML Datei verwendet werden können, die direkt im Visual Studio .NET bearbeitet werden kann. Der Sinn dieses früh angesetzten Meilensteins ist es, zeigen zu können, ob und wie mit dem Tool CodeSmith [5] die Ziele (namentlich die Codegenerierung für die HsrOrderApp_S4) erreicht werden können.

Meilenstein 2 Das Technologiestudium von CodeSmith, XML Schema und der für diese Arbeit relevanten Patterns ist abgeschlossen. Das Studium der Patterns bezieht sich hauptsächlich auf das Lesen von Büchern, insbesondere die Bücher von M. Fowler [1] und von E. Gamma [2]. Nicht in diesem Meilenstein enthalten ist das Technologiestudium der .NET Technologien, da es während der Arbeit immer andere Technologien

geben wird, in die man sich einarbeiten muss. Es gibt keine konkreten Resultate, die vorgewiesen werden. Der Meilenstein dient lediglich dazu, das nötige Basiswissen bis zu diesem Zeitpunkt zu erarbeiten.

Meilenstein 3 Die Analyse beinhaltet das Studium der bestehende HsrOrderApp_S3, die als Grundlage dieser Arbeit verwendet wird. Zudem wird eine Analyse gemacht, die zeigt, bei welchen Teilen einer Applikation es sinnvoll ist, generativ Code zu erzeugen und welche Patterns sich dafür eignen. Aus diesen Erkenntnissen sollen mögliche Vorschläge für Verbesserungen gemacht werden, die in die HsrOrderApp_S4 einfliessen.

Aus jeder der genannten Analysen entsteht eine Dokumentation, die mit diesem Meilenstein abgeschlossen werden soll.

Meilenstein 4 Aus der in Meilenstein 3 gewonnenen Erkenntnissen wird ein Design für die neu zu erstellende HsrOrderApp_S4 gemacht. Das Design wird mit diesem Meilenstein fertig gestellt.

Meilenstein 5 Die Implementation teilt sich zeitlich in 2 verschiedene Blöcke auf. Der erste Block wird mit dem Meilenstein 1 abgeschlossen, der zweite wird mit diesem Meilenstein abgeschlossen. Er umfasst die Implementation der HsrOrderApp_S4 und die Erstellung der CodeSmith Templates. Die HsrOrderApp_S4 ist eine Erweiterung der HsrOrderApp_S3 und soll mit Hilfe der erstellten Templates erzeugt werden können.

Meilenstein 6 Mit diesem Meilenstein wird die gesamte Dokumentation abgeschlossen. Vor allem beinhaltet dieser Meilenstein Dokumentation wie Management Summary, Abstract, Glossar, etc. Spezifische Teile der Dokumentation (wie z.B. eine Dokumentation zu CodeSmith) werden bereits in den korrespondierenden Phasen fertig gestellt und werden für diesen Meilenstein nur noch überarbeitet.

11.1.2 Vorgegebene Meilensteine

Zwischenpräsentation Die Zwischenpräsentation dient der Überprüfung der bis dahin erreichten Ziele. Um konkrete Resultate vorlegen zu können, sind die selbst definierten Meilensteine mit dieses Meilenstein

ab geglichen. Das bedeutet, dass bei der Zwischenpräsentation Meilenstein 1 bis 4 fertig gestellt wurden und bereits erste Resultate des Meilensteins 5 vorliegen sollten.

Abgabe Diplomarbeit Das eigentliche Ende der Diplomarbeit. An diesem Datum muss die Dokumentation und die Implementation vollständig abgeschlossen werden und die Resultate dem Betreuer abgegeben werden.

Ausstellung und Vorführung An diesem Tag findet eine öffentliche Ausstellung der Diplomarbeiten statt. Dazu müssen vorgängig Ausstellungs-Plakate erstellt werden und die Arbeitsplätze werden für die Vorführungen eingerichtet.

11.2 Erreichte Meilensteine

11.2.1 Meilenstein 1

Dieser Meilenstein wurde Zeitgerecht fertig gestellt. Es wurde eine XML Schema Beschreibung erstellt, aus der ein Codefile generiert wurde. Zudem konnte es ebenfalls direkt aus dem Visual Studio .NET ausgeführt werden.

11.2.2 Meilenstein 2

Dieser Meilenstein wurde ebenfalls zeitgerecht erreicht. Wobei hierzu zu sagen ist, dass dieser Meilenstein eher virtueller Natur ist. Konkrete Resultate wurden hier nicht geliefert. Trotzdem haben wir uns bis dahin mit den Grundlegenden Technologien vertraut gemacht, was vor allem das Studium der Patterns und von XML Schema beinhaltete.

11.2.3 Meilenstein 3

Dieser Meilenstein ist etwas vage formuliert und aus diesem Grund schwierig einzustufen. Trotzdem haben wir bis zu diesem Meilenstein dessen Ziele erreicht. Die Ergebnisse waren eher implizit, da wenig konkrete Arbeiten daraus resultieren. Es war evtl. nicht sinnvoll, diesen Meilenstein überhaupt zu definieren.

11.2.4 Meilenstein 4

Das Design wurde bis zu diesem Zeitpunkt grösstenteils fertiggestellt. Trotzdem haben wir auch nach diesem Meilenstein noch Änderungen vorgenommen. Der Grund dafür ist, dass wir während den Tests noch Unschönheiten am Design feststellten. Trotz dieser Änderungen erachten wir diesen Meilenstein als erreicht.

11.2.5 Meilenstein 5

Die Implementation hat sich herausgezögert. Dieser Termin war sehr optimistisch angelegt. Wir haben ihn leider verpasst und haben auch in der letzten Woche noch implementiert. Dieser Teil der Implementation bezog sich aber vor allem auf die Tests. Mit dem Templates und der HsrOrderApp_S4 waren wir zum Zeitpunkt dieses Meilensteins bereits fertig. Da unser Ziel aber war, die Implementationen vollständig abzuschliessen, erachten wir diesen Meilenstein als nicht erreicht.

11.2.6 Meilenstein 6

Die Abgabe wurde eingehalten und somit ist der Meilenstein erreicht.

11.3 Zeitaufstellung

Pro Woche sind $50h$ pro Person von der HSR vorgegeben. Dies entspricht einem totalen Pensum von $750h$. Die folgende Tabelle fasst die geschätzten und die tatsächlichen Stunden pro Kategorie zusammen. Die Stunden gelten jeweils als Total der von beiden Diplomanden aufgewendeten Stunden.

Tabelle 11.1: Soll / Ist Vergleich der aufgewendeten Stunden

Arbeit	**MS**	**Soll** $[h]$	**Ist** $[h]$
Allgemeines		**60**	**27**
• Meetings		10	9
• Präsentationen, etc.		30	9
• Installation SW / HW		20	9

Technologiestudium		**100**	**97**
• CodeSmith	2	20	16
• Patterns	2	40	54
• XML Schema	2	20	6
• .NET Technologien		20	21
Analyse		**100**	**92**
• HsrOrderApp_S3	3	40	35
• Anwendungen von generischen oder generativen Ansätzen	3	60	57
Design		**80**	**100**
• HsrOrderApp_S4	4	80	100
Implementation		**230**	**363**
• CodeSmith Prototyp	1	40	51
• Einbindung .NET Visual Studio	1	30	51
• Erstellung CodeSmith Templates	5	80	96
• HsrOrderApp_S4	5	80	140
• Testen			25
Dokumentation		**110**	**116**
• Project plan		10	15
• Allgemein (Vision, Glossar, etc.)	6	60	50
• CodeSmith		30	30
• Abstract, Management Summary, etc.	6	30	21
Total		**700**	795

11.4 Sitzungen

11.4.1 Sitzung mit Betreuer

Wann: Wöchentlich, normalerweise montags.
Wer: H. Huser, M. Egli, M. Winiger
Was: Wochenziele definieren, Probleme besprechen.

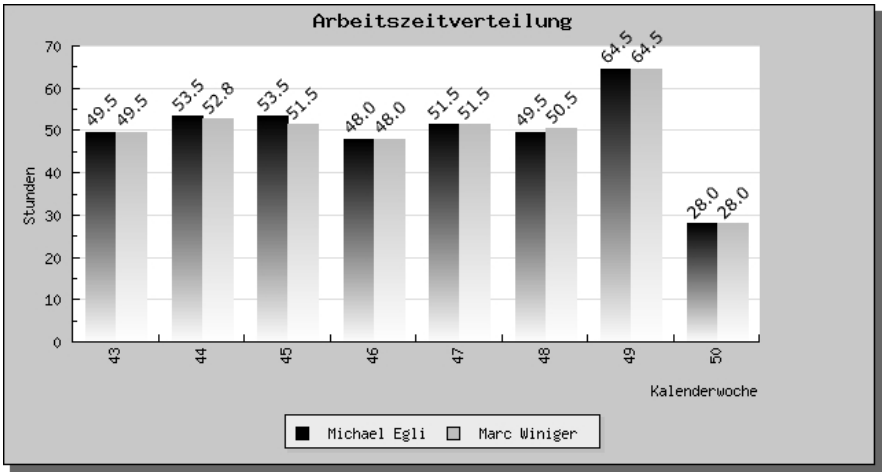


Abbildung 11.1: Zeiterfassung

11.4.2 Teamsitzung

Wann: Täglich um 8.00 Uhr
Wer: M. Egli, M. Winiger
Was: Tagesziel definieren.

Kapitel 12

Ressourcen

Tabelle 12.1: Benutzte Resourcen und Tools

Hardware	2 standard Siemens Scenic PC's (von der HSR zur Verfügung gestellt)
OS	Standard Windows 2000 Installation der HSR
Raum	1.262
Entwicklung	• Microsoft Visual Studio .NET 2003 • Codesmith 2.1.1270 [1] • Microsoft SQL Server 2000 • Internet Information Services (IIS)
Dokumentation	• MiKTeX 2.3 [2] • TeXnicCenter 1 Beta 6.01 [3]
Versionierung	• Concurrent Version System (CVS) 1.11.6 [4] • TortoiseCVS 1.4.5 [5]

[1] http://www.ericjsmith.net/codesmith/
[2] http://www.miktex.org/
[3] http://www.toolscenter.org/products/texniccenter/
[4] http://cvshome.org/
[5] http://www.tortoisecvs.org/

Teil V

Testbericht

Kapitel 13

Ausgangslage

Getestet wurde die Funktionalität der HsrOrderApp_S4. Da diese Applikation alle unterstützten Fälle der Code Generierung abdeckt, ist damit auch die Richtigkeit der Templates gewährleistet.

13.1 Testumgebung

Die Test wurden lokal ausgeführt und damit ist die Testumgebung dieselbe, wie in Kapitel 12 in der Tabelle 12.1 angegeben wurde.

13.2 Datenbank

Das Design der Datenbank wurde vorgegeben. Es handelt sich dabei um die Datenbank für die HsrOrderApp. Diese wurde für die in dieser Arbeit erstellten Referenzimplementation verwendet und dient auch als Datenbank für die Tests.

Kapitel 14

HsrOrderApp_S4

Wir haben entschieden, die bestehenden Tests der HsrOrderApp_S3 zu erweitern. Dazu haben wir Facade, WebServices, TestProxy und TestApplikation übernommen und an unsere Architektur angepasst.

Getestet wurde damit der generierte Code der Domain Object und der Mapper.

14.1 Anforderungen an die Applikation

- Neue Objekte

  Objekte über die Factory erstellen und in der UnitOfWork als new markieren.

- Objekte ändern

  Geladene Objekte verändern und in der UnitOfWork als dirty markieren.

- Objekte in die Datenbank schreiben

  Neue Objekte in die Datenbank einfügen und geänderte Objekte in der Datenbank auf den neusten Stand bringen.

- Objekte aus der Datenbank lesen

  Die Daten für neu zu erstellende Objekte werden mittels find Methoden aus der Datenbank geholt.

- Objekte als gelöscht markieren

  Objekte für das löschen in der Datenbank bei der UnitOfWork registrieren.

- Rekursives Löschen

  Wird ein Objekt mit ToMany Relationen gelöscht, müssen alle zugeordneten Objekte vorher auch gelöscht werden.

- Werfen von Concurrency Exceptions

  Werden versucht veraltete Objekte auf die Datenbank zu schreiben, muss eine Concurrency Exception geworfen werden.

14.2 Durchgeführte Tests

Die folgenden drei Test werden mehrmals hintereinander ausgeführt. Zuerst werden jeweils die in der letzen Iteration erstellten Objekte gesucht und wenn vorhanden gelöscht. Objekte einer ToMany Relation müssen von den Mappern automatisch gelöscht werden. Das dies auch wirklich geschieht, wird durch Constraints auf der Datenbank überprüft.

14.2.1 Person Test

Es werden drei neue Personen Objekte erstellt. Zu jeder Person werden zwei Adressen erstellt. Diese werden in die Datenbank geschrieben.

Danach werden die Personen Objekte geändert, je eine zusätzliche Adresse hinzugefügt und ebenfalls in die Datenbank geschrieben.

14.2.2 Product Test

Es werden 16 Produkte Objekte in vier Kategorien erstellt und auf die Datenbank geschrieben.

Davon werden 12 Objekte verändert und 4 Objekte wieder gelöscht. Diese Operationen werden in einem Schritt auf der Datenbank ausgeführt.

14.2.3 Order Test

Die Personen und Produkte aus den vorherigen Tests werden über die Finder Methoden gesucht.

Der ersten Person werden drei Bestellungen mit je drei Positionen hinzugefügt. Diese Änderungen werden auf die Datenbank geschrieben. Danach werden die Positionen geändert, hinzugefügt und gelöscht.

14.3 Ergebnisse

Die Tests laufen ohne Fehler durch. Der Trace vom SQL Profiler und eine Kontrolle in der Datenbank belegen, dass die Einträge auch wirklich geschrieben wurden.

Werden während den laufenden Tests die Datenbank-Einträge verändert, sollte eine Concurrency Exception geworfen werden. Die Applikation stürzt jedoch einer NullReferenzeException ab. Nach beheben dieses Fehlers reagiert alles wie erwartet.

Anhang A

Persönliche Berichte

A.1 Michael Egli

Im Nachhinein kann muss ich sagen, dass dieses Projekt in vielen Hinsichten sehr spannend war. Zu Beginn war ich zwar etwas skeptisch, das hat sich aber schon bald in Interesse für das Projekt gewandelt. Das Spannende am Projekt für mich war, dass es sehr viele technische Aspekte tangierte. Zum Einen war dies XML und XML Schema, zum Anderen aber auch generische und generative Programmierung. Ebenfalls spannend war die pattern-orientierte Entwicklung. Für mich war es das erste Projekt, bei dem ich so intensiv Patterns verwendet habe.

Leider muss man sagen, dass wir sehr viele Stunden mit CodeSmith verbracht haben. Das Tool ist sehr schlecht dokumentiert und es funktioniert auch nicht alles, wie wir es erwartet haben.

Für mich ist das Resultat der Arbeit sehr erfreulich. Ich glaube wir können mit *STORM* einen guten Lösungsansatz für das gestellte Problem präsentieren. Natürlich wäre es ausbaufähig, aber die Zeit war leider sehr knapp dafür.

Das Projekt war über das Ganze gesehen sehr denklastig, was ich als positiv bewerte. Wir mussten uns sehr oft genau überlegen, wie wir ein Problem lösen können.

Die Zusammenarbeit mit Marc kannte ich schon aus der letzten Semesterarbeit. Obwohl die Zeit während der Diplomarbeit sehr anstrengend war, hat die Teamarbeit immer funktioniert. Ich denke, dass wir uns sehr gut

ergänzt haben und auch, dass wir es geschafft haben die Stärken von beiden gut auszunutzen.

Die Betreuung fand ich sehr gut. Zum Einen hatten wir Betreuung von Herrn Huser, zum Anderen auch noch von M. Konrad. Dies hat mich positiv überrascht und hat uns auch sehr oft geholfen. Es wäre schön, wenn es immer so wäre, aber leider haben wir in früheren Projekten auch schon Anderes erlebt.

Zusammenfassend kann ich sagen, dass ich sehr viel gelernt habe. Vor allem auch Dinge wie generative Programmierung, die ich unter Umständen in Zukunft gut gebrauchen kann.

A.2 Marc Winiger

Kurz vor Beginn der Diplomarbeit wurde das ursprünglich geplante Thema verworfen. Das neue Thema war generative Programmierung. Zuerst war ich eher skeptisch, denn bisher habe ich immer wenn ich etwas Entwickelt habe, versucht das Problem so generisch wie Möglich zu lösen. Die generative Programmierung war für mich also ein ganz neuer Aspekt.

Es ist nicht einfach sich in Code einzuarbeiten, der von jemand anderem entwickelt wurde. Das Ziel war es die existierende HsrOrderApp umzubauen und Teile davon mit generiertem Code zu ersetzen. Beim Ändern einer existierenden Applikation, kann nach unseren Erfahrungen sehr viel Zeit verloren gehen. Nach den negativen Erfahrungen der letzen Semesterarbeit, bin ich froh, dass wir die Möglichkeit hatten, ein eigenes Projekt zu starten und zu einem späteren Zeitpunkt die noch fehlenden Teile von der existierenden Applikation zu übernehmen. So hatten wir relativ schnell einen Funktionierenden Prototypen.

Im Laufe des Projekt habe ich immer mehr Gefallen an der gerativen Programmierung gefunden. Denn es ist zwar ein grosser Aufwand ein Template zu schreiben, das alle Fälle abdeckt und auch noch fehlerfreien Code erzeugt. Doch wenn man das erst mal gemacht hat, ist es genial, wenn man nur noch eine einfache Beschreibung braucht aus denen dann hunderte Zeilen Code generiert werden.

Obwohl wir am Anfang mit dem Lösungsansatz XML Schema theoretisch etwas Zeit verloren haben, habe ich es interessant gefunden verschiedene Wege zu sehen, wie die Generierung des Codes konfiguriert werden kann.

Nachdem ich bereits bei der letzen Semesterarbeit mit Michael zusam-

men gearbeitet habe, war für mich klar, dass ich auch die Diplomarbeit mit ihm durchstehen möchte. Es sind acht Wochen, in denen man sehr viel Zeit miteinander verbringt. Trotzdem hatten wir nie ernsthafte Auseinandersetzungen. Ich würde mit ihm sofort wieder ein Projekt angehen.

Es waren acht doch ziemlich strenge Wochen auf die wir zurückblicken können. Doch da wir von Anfang ein Ziel vor Augen hatten, war es immer möglich, sich irgendwie zu motivieren. Alles in allem muss ich sagen dass es eine schöne, interessante und auch lehrreiche Zeit war.

Anhang B

Protokolle

B.1 Sitzung 1 vom 20. Oktober 2003

B.1.1 Teilnehmer

H. Huser (hhu), M. Egli (meg), M. Winiger (mwi)

B.1.2 Themen

- Besprechung Aufgabenstellung
- Allgemeine Fragen zum Projektstart

B.1.3 Aufgabenstellung

Die Aufgabenstellung ist noch nicht fix. Es ist möglich, dass im Verlauf der Arbeit noch Verschiebungen der erwarteten Resultate möglich sind.

B.1.4 Entscheidungen

- Der Projektplan muss versioniert werden, damit Änderungen während dem Projekt auch im Nachhinein ersichtlich sind.

- Wir setzen eine Online-Zeiterfassung ein. Die Beschreibung zu den Einträgen dient gleichzeitig als Projekt-Tagebuch.
  http://homer.skymx.net/zeit/ooxgen/
- Wir würden die Dokumentation gerne mit LaTeX erstellen. Wir versuchen eine Lösung zu finden, dass wir die Dokumentation, bzw. einzelne Dokumente davon, in ein zu Microsoft Word kompatibles Format umwandeln können.
- Einzelne Dokumente (z.B. Technischer Bericht) werden in Englisch verfasst.

B.1.5 Hinweise

- (hhu): Auf der MSDN Seite finden wir Pattern von Microsoft, die für uns nützlich sein könnten.
  http://msdn.microsoft.com/architecture/patterns/MSpatterns/
  http://msdn.microsoft.com/architecture/application/default.aspx?pull=/library/en-us/dnbda/html/distapp.asp
- (hhu): Der erwartete Einsatz für die Diplomarbeit liegt bei 50h/Woche.

B.1.6 Ziele

- Erstellen des Projektplans, damit dieser an der nächsten Sitzung besprochen werden kann.
- Einarbeiten in HsrOrderApp_S3
- Vertiefung von Pattern und .NET Technologien

B.2 Sitzung 2 vom 28. Oktober 2003

B.2.1 Teilnehmer

H. Huser (hhu), M. Egli (meg), M. Winiger (mwi)

B.2.2 Themen

- Wochenrückblick
- Projektplan v1.0
- Dokumentation

B.2.3 Wochenrückblick

- Der Projektplan (v0.1) liegt vor und die Meilensteine sind in der Doku beschrieben.
- Die Technologiestudien Patterns und CodeSmith sind grösstenteils abgeschlossen.

B.2.4 Projektplan v0.1

Der Projektplan ist soweit in Ordnung. Die Planung ist für uns nicht einfach, da wir uns zuerst mit den Technologien und Patterns vertraut machen müssen. Wichtig ist, dass wir auftretende Änderungen am Projektplan sofort mit (hhu) besprechen.

B.2.5 Dokumentation

Risikoanalyse Zu allgemein. Konkretere Lösungen müssen angegeben werden.

Meilenstein 3 Das Ziel ist zu hoch angesetzt. Es reicht wenn wir zeigen welche Art von Mappings durch generativen Code ersetzt werden können.

Implementation Die Erstellung der Templates, welche als Werkzeuge eingesetzt werden können, sollten in der Planung von der Implementation der Beispiel-Applikation (HsrOrderApp_S4) getrennt werden.

Bibliographie Die MSDN Architecture Patterns von Microsoft sollten in die Bibliographie aufgenommen werden.

Allgemein Wir generieren aus dem Technical Report ein eigenständiges Dokument.

B.3 Sitzung 3 vom 3. November 2003

B.3.1 Teilnehmer

H. Huser (hhu), M. Egli (meg), M. Winiger (mwi)

B.3.2 Themen

- Anforderungs-Spezifikation
- Technischer Bericht
- Weiteres Vorgehen

B.3.3 Anforderungs-Spezifikation

Q: Es gibt unsererseits Unklarheiten bezüglich dem Verfassen einer Anforderungs-Spezifikation.
A: Vor allem im ersten Teil der Arbeit geht es darum die bestehende Applikation zu analysieren. Wir müssen Lösungsansätze suchen und herausfinden was die beste Lösung ist. Darum werden sich Anforderungen erst im Verlaufe des Projekt ergeben.

B.3.4 Technischer Bericht

Der erste Teil des technischen Berichts liegt vor. Es handelt sich dabei um eine Zusammenfassung der für unser Projekt wichtigen Pattern aus dem Buch von Fowler [1] und einer kurzen Beschreibung zu CodeSmith [5].

Wir werden das PDF auf `http://ooxgen.linda.homelinux.net/` laden und würden gerne an der nächsten Sitzung ein Feedback erhalten.

B.3.5 Weiteres Vorgehen

Die Analyse der bestehenden Applikation, die Erstellung von Templates und das Dokumentieren der Erkenntnisse erfolgt im Moment fast gleichzeitig. Anfangs Woche werden wir den Schwerpunkt auf die Spezifikation der XML Schemas und das programmieren des zum parsen benötigten Code setzen.

B.4 Sitzung 4 vom 12. November 2003

B.4.1 Teilnehmer

H. Huser (hhu), M. Egli (meg), M. Winiger (mwi)

B.4.2 Themen

- Lebensdauer von Objekten
- Doku
- Planung/Fortschritt

B.4.3 Lebensdauer von Objekten

Es müssen verschiedene Fragen bezüglich Lebensdauer der Objekte angeschaut werden.

Ist es besser wenn die Objekte nach einem Commit zerstört werden, oder können sie evtl. später wieder verwendet werden? Wird die Konsitenz bei Objekten aus der IdentitiyMap überprüft? Können die Objekte eingeteilt werden in eher statische Daten und solche, die man jedesmal neu von der Datenbank holen muss? Soll die Wahl zwischen Geschwindigkeit und Aktualität dem Benutzer überlassen werden?

B.4.4 Doku

Wir haben die beiden Technologien XML Schema und Reflection im Vergleich dokumentiert. Wir müssen den Teil noch überarbeiten und werden ihn vor der nächsten Sitzung mailen.

B.4.5 Planung/Fortschritt

CodeSmith bereitet uns immer wieder Schwierigkeiten, weil vieles nicht gerade auf Anhieb funktioniert. Wir haben etwas früher mit der Implementation einer neuen HsrOrderApp begonnen und sind jetzt an den ersten Versuchen die Templates einzubauen.

B.5 Sitzung 5 vom 18. November 2003

B.5.1 Teilnehmer

H. Huser (hhu), M. Egli (meg), M. Winiger (mwi)

B.5.2 Themen

- Generischer/generativer Ansatz
- Enums, Lookup-Tables

B.5.3 Generischer/generativer Ansatz

Es zeigt sich, dass der generative Ansatz viel aufwendiger zu realisieren ist, als am Anfang angenommen. Es gibt sehr viele Spezialfälle. Eine Applikation wie die HsrOrderApp ist zu klein, dass sich der ernsthafte Einsatz von Code-Generierung lohnen würde.

B.5.4 Enums, Lookup-Tables

Enums sind einer dieser Spezialfälle. Es gibt verschiedene Möglichkeiten diese auf die Datenbank zu mappen. Eine Möglichkeit wären Lookup-Tables. Dabei werden die Enum-Beschreibungen, mit eindeutigen IDs, in einzelnen oder einer gemeinsamen Tabelle abgelegt.

Eine entscheidende Rolle spielt für den Mapping-Code, wie die Enums in der Datenbank abgelegt sind. Liegt ein unveränderbares Datenbank-Schema vor, muss der Code mit grosser Wahrscheinlichkeit spezifisch geschrieben werden.

Abgesehen davon werden die Enums in der HsrOderApp-Datenbank nicht sehr schön abgelegt. Sie sind nicht vollständig normalisiert und somit redundant als VARCHAR abgelegt, was das mapping noch ein Stück komplizierter macht

B.6 Sitzung 6 vom 24. November 2003

B.6.1 Teilnehmer

H. Huser (hhu), M. Egli (meg), M. Winiger (mwi)

B.6.2 Themen

- Projektstand
- Zwischenpräsentation

B.6.3 Projektstand

Domain-Objekte können bereits kompilierbar mit den Templates generiert werden. Die Templates für die Mapper sollten auch bald soweit sein.

Die Templates sind so geschrieben, dass Primary Keys auch über mehrere Kolonnen definiert sein können. Das hat zum Teil etwas mehr Zeit benötigt als erwartet. Es wäre einfacher gewesen nur Surrogate Keys zu behandeln. Wir haben jetzt jedoch schon überall mehrteilige Keys verwendet.

B.6.4 Zwischenpräsentation

Am Mittwoch 26. November um 15.00 Uhr findet eine Sitzung mit Hr. Bärfuss statt. Wir erklären den verwendeten Ansatz mit Attributierung und Reflection und präsentieren was wir bisher gemacht haben. Ausserdem sollten wir unsere Erkenntnisse aufzeigen.

Wir erstellen einen aktualisierten Projektplan bis Mittwoch.

Literaturverzeichnis

[1] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2003. ISBN 0-321-12742-0. 560 pp.

[2] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements od Reusable Object-Oriented Software*. Addison-Wesley Publishing Company, New York, 1997. ISBN 0-201-63361-2. 416 pp.

[3] University Linz and ETH Zürich. Coco/R, compiler generation tool for C#. URL http://dotnet.jku.at/projects/rotor/.

[4] Microsoft. Microsoft patterns and best practices. URL http://msdn.microsoft.com/architecture/patterns/MSpatterns/.

[5] Eric J. Smith. CodeSmith Homepage. URL http://www.ericjsmith.net/codesmith/.