

Gunther Dörnen und Stefan Waltenspiel, wega Informatik AG

# Generierung der Datenbank Zugriffsschicht mit NVelocity für das .NET Framework und dem Oracle Data Provider (ODP)

Wer eine Oracle-Datenbank aus dem .NET Framework heraus ansprechen möchte hat aufgrund der geringeren Integration von .NET in Oracle Datenbanken einiges an Handarbeit zu leisten. Eine der Zugriffsmöglichkeiten auf Datenbanken in .NET ist ADO.NET. Eine grosse Lücke in ADO.NET liegt in einer automatisierten Persistenzunterstützung der .NET Datasets gegen Oracle Datenbanken. Es gibt nur eine Ansammlung von Klassen der Datenbankschnittstelle, die z.B. durch den ODP implementiert werden und die viel Handarbeit erfordern.

In diesem Artikel wird die Erfahrung der wega Informatik AG mit einer Lösung für ADO.NET vorgestellt, die dieses Problem beseitigt. Das hier vorgestellte ausgereifte Konzept durfte sich im Rahmen eines Grossprojektes bewähren. Verwendet wurde in der Lösung die Template Engine NVelocity (<http://sourceforge.net/projects/nvelocity/>), eine Portierung des populären Java projects Velocity in das .NET Framework.

## Der Lösungsansatz für Oracle Datenbanken

Als Tool für die automatische Generierung der Datenzugriffsschicht dient ein selbst entwickeltes wizard basierendes Werkzeug, welches im Sinne eines «reverse engineering» über die vorhandene Datenbank die Persistenzklassen und mehrere DataSet Schemata (typed DataSet's) generiert. Der in C# implementierte Wizard nutzt die API der NVelocity Template Engine und unterstützt:

- Die Definition der DataSet's pro Use Case. Nur der Name muss angegeben werden.
- Die Auswahl der gewünschten Entitäten, sowie die Zuordnung optionaler Sequenzen.
- Die Zuordnung der ausgewählten Entitäten zu einem oder mehreren der definierten DataSet's.

## Das Ergebnis

Architektur der Persistenzschicht

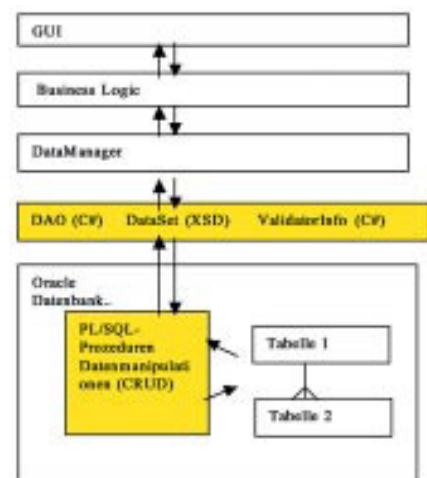


Figure 2: Übersicht Architektur. Die gelb markierten Teile werden durch den hier vorgestellten Code-Generierungsmechanismus generiert.

Bei der Implementierung von verschiedenen Use Cases in einem Projekt werden individuell strukturierte Datencontainer benötigt. Als eines der Resultate der Generierung (aus Oracle Metadaten und Template) bekommt der Entwickler deshalb ein DataSet Schema generiert, welches die definierte Menge von Entitäten, ihre Schlüssel und ihre darauf basierenden Beziehungen untereinander beschreibt.

Mit Hilfe der Entwicklungsumgebung (MS Visual Studio) wird aus diesem XSD-Schema ein «typed DataSet», d.h. eine C# Klasse, die in der Implementierung der Use Cases als Datencontainer zur Verfügung steht.

Als Bausteine der automatisierten Persistenzschicht erhält der Entwickler pro Entität ein «Data Access Object»

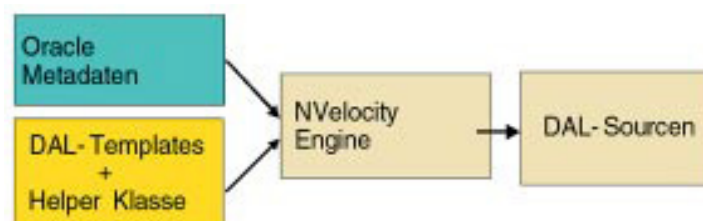


Figure 1: Übersicht Sourcegenerierung mit NVelocity. Der Gelb markierte Teil wurden von wega erstellt (DAL = Data Access Layer).

(kurz DAO – Klasse), welche die Daten-selektion und Datenmanipulation über individuelle Methoden ermöglichen. Die DAO-Klassen werden nach Bedarf im DataManager des UseCases kombiniert.

Basis der DAO-Klassen ist (für den «Use Case Implementer» uninteressant, weil durch die DAO-Klasse gekapselt) ein ebenfalls generiertes PL/SQL Package. In den Stored Procedures sind die verschiedenen möglichen SELECT, INSERT, UPDATE und DELETE Statements implementiert.

Zusätzlich wurden C# Validierungsklassen hinzugefügt, die vor Datenmanipulationen die DataSet's auf NOT NULL Constraints, Zeichenkettenlängen etc. überprüfen können.

Grundsätzliches Ziel des Konzeptes war es, Sourcen zu generieren die nicht modifiziert werden dürfen. Für Spezialfälle, also spezielle Datenbankzugriffe wird parallel ein einfaches PL/SQL Package mit einer zugehörigen nicht generierten DAO-Klasse für manuelle Zugriffe geführt.

Lediglich im Falle der DataSet's wurden bisher manuelle Erweiterungen ausgeführt. Aber einzig aus dem

Grund, als es neben der Oracle Datenbank weitere Persistenz Systeme gibt. Eine Trennung wäre hier nur schwer implementierbar gewesen. Ansonsten konnte aber von manuellen Eingriffen abgesehen werden.

## Velocity und Templates

Als Basis für die Source Generierung dient die in C# implementierte Wizard Applikation.

Über die Velocity API werden Templates ausgeführt, welche mit der Velocity Template Language (VTL) geschrieben wurden. Mit VTL wiederum können eigene C# Klassen als Helper (nicht zu verwechseln mit der OracleHelper Klasse) integriert werden, indem sie im Velocity Ausführungskontext den Templates zur Verfügung gestellt werden.

Templates enthalten Text und VTL Elemente. Der Text wird unmittelbar ausgegeben, die VTL Anweisungen werden vom Interpreter ausgeführt und führen am Ende auch zu Ausgabeanweisungen. Eine grobe Übersicht der wesentlichen Elemente zeigt Figure 3.

Mit VTL wird eine einfache aber mächtige Scriptsprache zur Verfügung gestellt. Ziel ist eine Transformation des Templates in die gewünschte Ausgabeform. Die zentrale Steuerung der Transformation obliegt in diesem Fall den Oracle Metadaten. Wer Erfahrung mit XSLT aus der XML-Welt hat, kann hier gewisse Parallelen erkennen.

Die Stärke des Werkzeugs ist die Integration in das .NET Framework. Mit NVelocity liegt eine ausreichend stabile Portierung des Originals aus der Java Welt vor. VTL ermöglicht den Zugriff auf eigene .NET Hilfsklassen via Reflection. Dies ist erforderlich, da insbesondere die verwendeten Oracle Metadaten für die individuellen Transformationen und unterschiedlichen Templates sinnvoll aufbereitet werden müssen.

## Die Templates der Persistenzschicht

Für die Persistenzschicht wurden folgende Templates erarbeitet:

- DataSet Schema Template (XSD-Schema)
- Template für DAO-Klasse (C#)

- Template für Stored Procedures (PL/SQL Package)
- Template für Validierung (C#)

Wie die Aufzählung zeigt, werden Sourcen für 3 verschiedene Sprachen generiert, die am Ende zusammen harmonisieren.

Obwohl die Entwicklungsumgebung «MS Visual Studio» die DataSet Erstellung visuell unterstützt, wurde dieses Feature bei der Generierung verzichtet. Die Entwicklung eines DataSet Templates stellte sich als relativ einfach heraus und ermöglichte so die Integration in das Gesamtkonzept.

Im Figure 3 wird das Konzept hinter den Templates sichtbar. VTL-Elemente werden durch den Literal # markiert. Alles andere wird als Text unverändert ausgegeben.

Als Grundlage für das Template dient ein XSD-Schema, das um VTL Anweisungen erweitert wurde.

Nachfolgend einige Erläuterungen zum Figure 3 und darin enthaltenen VTL-Anweisungen:

- ## Kommentar
- ## Anweisungen unter einer Bedingung  
#if (<condition>) ... #end

- ## Iteration über alle Entitäten. Diese werden in der Variablen tableNameRow zur Verfügung gestellt. #foreach (\$tableNameRow in \${SchemaDataSet.Entity.Rows})

Mit der Variablen «SchemaDataSet» wird auf das DataSet der Oracle Metadaten zugegriffen, welches im Velocity-Ausführungskontext dem Template direkt zur Verfügung gestellt wird.

```
##=====//
## -----
## Velocity Template for DataSet generator
## -----
##
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="{dataset_name}" targetNamespace="http://tempuri.org/{dataset_name}.xsd"
elementFormDefault="qualified" attributeFormDefault="qualified" xmlns="http://tempuri.org/{dataset_name}.xsd"
xmlns:mstns="http://tempuri.org/{dataset_name}.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="{dataset_name}" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
##
## Write out Table definition
##
#foreach ($tableNameRow in ${SchemaDataSet.Entity.Rows})
#if ($helper.IsTableForProcess($tableNameRow.IsSelected, $IsMaster, $tableNameRow.Name, $dataset_name,
$SchemaDataSet))
  <xs:element name="{tableNameRow.Name}">
    <xs:complexType>
      <xs:sequence>
.....
```

Figure 3: Auszug aus dem DataSet Template

```
...
-- =====
-- SELECT PROCEDURES FOR TABLE ${tableNameRow.Name}
-- Select by Primary Key
-- =====
procedure $helper.UniqueName(${tableNameRow.Name})_SELPK_P
(result OUT retCursor
#foreach($primaryKeyColumn in $helper.GetPrimaryKeyColumns(${tableNameRow.Name},
$SchemaDataSet))
, v$primaryKeyColumn.ColumnName IN
${tableNameRow.Name}.${primaryKeyColumn.ColumnName}%TYPE
#end
)
#if ($body == "true")
is
begin
open result for
select
#ColumnEnumeration("")
from ${tableNameRow.Name} where
...
```

Figure 4: Auszug aus dem PL/SQL Package Template

## Die Aufgabe der Oracle Metadaten

Aus dem reichhaltigen Oracle-Metadatenpool werden alle Informationen für die Codegenerierung den NVelocity-Templates zur Verfügung gestellt.

Aus den Beschreibungen der Tabellenattribute werden z.B. «xs:attribute» Elemente im Schema oder «Oracle-Parameter» Instanzierungen in der C# DAO-Klasse (mit der erforderlichen Initialisierung von Typ, Länge und Attributnamen). Im PL/SQL Package werden aus dem Tabellenattribut Metadaten Signaturen für die verschiedenen PL/SQL Prozeduren oder «ref cursor select statements» generiert.

Aus den Tabellen Metadaten werden «xs:elemente» im Schema oder Klassennamen für die DAO-Klassen.

Beziehungen zwischen Entitäten werden in «xs:key» und «xs:keyref» Elemente im Schema transformiert.

Dies ermöglicht dem «Use Case Implementer» typisierte DataRow's z.B. in einer Parent-Child Beziehung zu verknüpfen, ohne das er sich um die zugrundeliegende Beziehung kümmern muss.

Informationen über Indexe werden in DAO-Select Methoden überführt, die wiederum auf die selbstverständlich auch generierten zugehörigen PL/SQL Prozeduren mit den statischen «ref cursor select statements» gemappt werden.

Für die Fälle, die von den index-basierenden statischen und performanten Zugriffsmethoden nicht abgedeckt werden, steht die allgemeine Select-Methode zur Verfügung, an die als Übergabeparameter eine beliebige «where-clause» übergeben werden kann.

*Ein Dankeschön gilt an dieser Stelle meinem irischen Teamkollegen James Dooley, der hier in Vorarbeit ein DataSet für die Metadaten samt füllender Klasse als Basis zur Verfügung stellen konnte.*

## OracleHelper Klasse

Die Aufgabe der OracleHelper Klasse ist eine erste Stufe der Vereinfachung der DB-Zugriffe für den Oracle Data Provider. Verwendet wird die OracleHelper Klasse ausschließlich durch die DAO-Klassen. D.h. die Integration erfolgt im DAO-Template. Die OracleHelper Klasse wurde in Verbindung mit umfangreichen NUnit-Tests entwickelt, weiterentwickelt und geprüft. In ihr ist z.B. durch Event-Handling ein Konzept für «optimistic concurrency check» integriert. Das getestete Zugriffskonzept hatte wiederum Einfluss auf die Entwicklung der DAO-Templates.

## Vorteile der vorgeschlagenen Lösung

Mit dieser Eigenentwicklung eines Persistenzframeworks für DataSet's konnte eine entscheidende Lücke in ADO.NET hinsichtlich der Oracle Unterstützung geschlossen werden.

Es bietet folgende Vorteile

- **Zeitersparnis (=Kosten)**
- **Einfach durch jedermann modifizierbar und erweiterbar**

## Weitere Informationen

- wega Informatik AG  
<http://www.wega-informatik.ch>
- Der Oracle Data Provider für .NET  
<http://otn.oracle.com/tech/windows/odpnet/index.html>
- Das Original: Velocity bei Apache/Jakarta mit Details zur Sprache VTL  
<http://jakarta.apache.org/velocity/>
- Die .NET Portierung von Velocity  
<http://nvelocity.sourceforge.net/>

Individuelle kundenspezifische Anforderungen hinsichtlich «optimistic concurrency check» oder die Implementierung eines statusbasierenden «delete» in Abhängigkeit vom Vorhandensein eines spezifischen Attributes («ROW\_STATUS\_CODE») konnten leicht implementiert werden und können auch leicht an andere Kundenbedürfnisse angepasst werden. Denkbar sind weitere Templates für die Generierung von «NUnit Testcases».

- **Automatisierte Validierung (NOT\_NULL, Attributlängen etc.)**

- **Qualität**

- **Performanz**

Durch das Endergebnis wird ein «Use Case Implementer» sinnvoll unterstützt. Ein robustes ADO.NET DataLayer mit dem generierten Persistenzframework auf die Beine zu stellen ist hiermit zur leichten Übung geworden.

## Contact

wega Informatik AG

Gunther Dörnen  
Stefan Waltenspiel

E-Mail:  
gunther.doernen@wega-informatik.ch  
stefan.waltenspiel@wega-informatik.ch