



Oracle Designer – Nutzung in einem Langläufer-Projekt

Rolf Wesp . Consultant . 13.03.2008

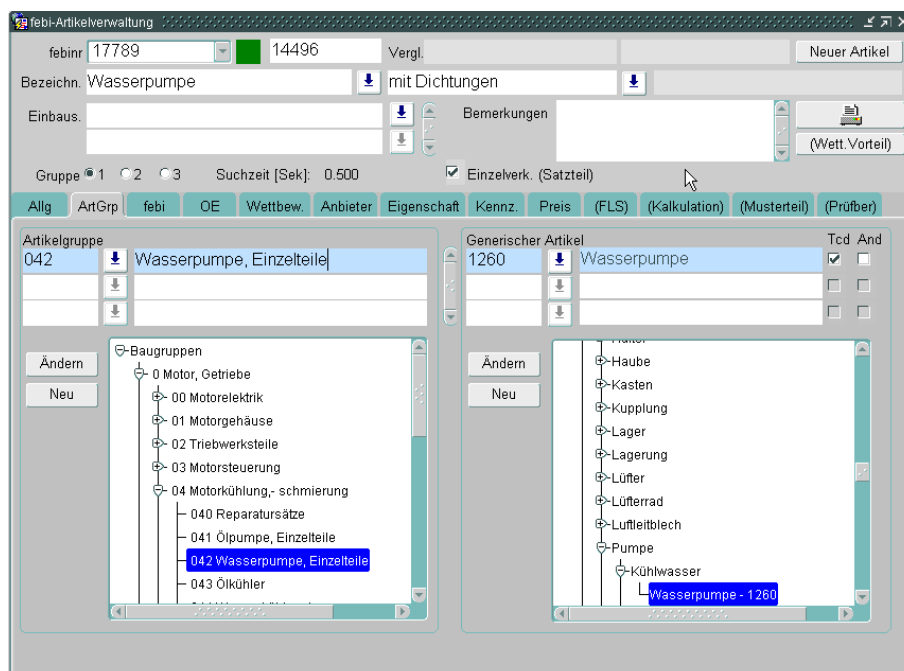
Allem Anschein nach hat der Oracle Designer seine Zukunft bereits hinter sich.

Oracle erklärt, dass der Support für den Designer an den von Forms gebunden ist – mindestens bis 2013, darüber hinaus gibt es keine Aussage. Dem steht weiter ein reges Interesse innerhalb der Community an dem Produkt gegenüber. Hier will man nicht nur sichergestellt wissen, dass die im Repository gebundenen Investitionen geschützt sind, sondern erwartet auch eine Weiterentwicklung, die mindestens das Nachführen neuer Datenbank-Features in die Modellierungs- und Generierungs-Tools umfasst.

Der folgende Beitrag zeigt, auf welche Weise der Designer in einem über mehrere Jahre laufenden Projekt genutzt wurde und will dessen Bedeutung für die strukturierte und effiziente Weiterentwicklung hervorheben.

1. Der Anwender und das Projekt

Zweck der Fa. Ferdinand Bilstein GmbH & Co. KG, Ennepetal, ist die Produktion und der Vertrieb von Ersatzteilen im so genannten Automotive Aftermarket. Seit dem Jahr 2000 arbeite ich in einem Projekt am Aufbau und der Weiterentwicklung eines Informationssystems, dessen Kern die Verwaltung von Artikeldaten und Artikelverwendungen in PKW, NKW und Anhängern bildet. Aus kleinen Anfängen ist im Laufe der Zeit eine Anwendung von beachtlichem Umfang und hoher Komplexität gewachsen.



Rund um den Kern, der kontinuierlich funktional erweitert und an neue Anforderungen angepasst wird, die sich u. a. aus der Integration mehrerer Tochterunternehmen ergeben haben, ist eine Reihe weiterer Module entstanden, welche beispielsweise der Bewertung von Marktchancen, der Identifikation neuer Artikel und der Vervollständigung und Differenzierung der Artikeldaten dienen.

Abbildung 1: Artikelverwaltung

Komplettiert wird das System durch die automatisierte Bereitstellung von Schnittstellendateien für externe Weiterverarbeitungen (CDs verschiedener Provider) und die Replikation gegen eine Datenbank, die den Artikelkatalog im Web darstellt (www.febi.com).

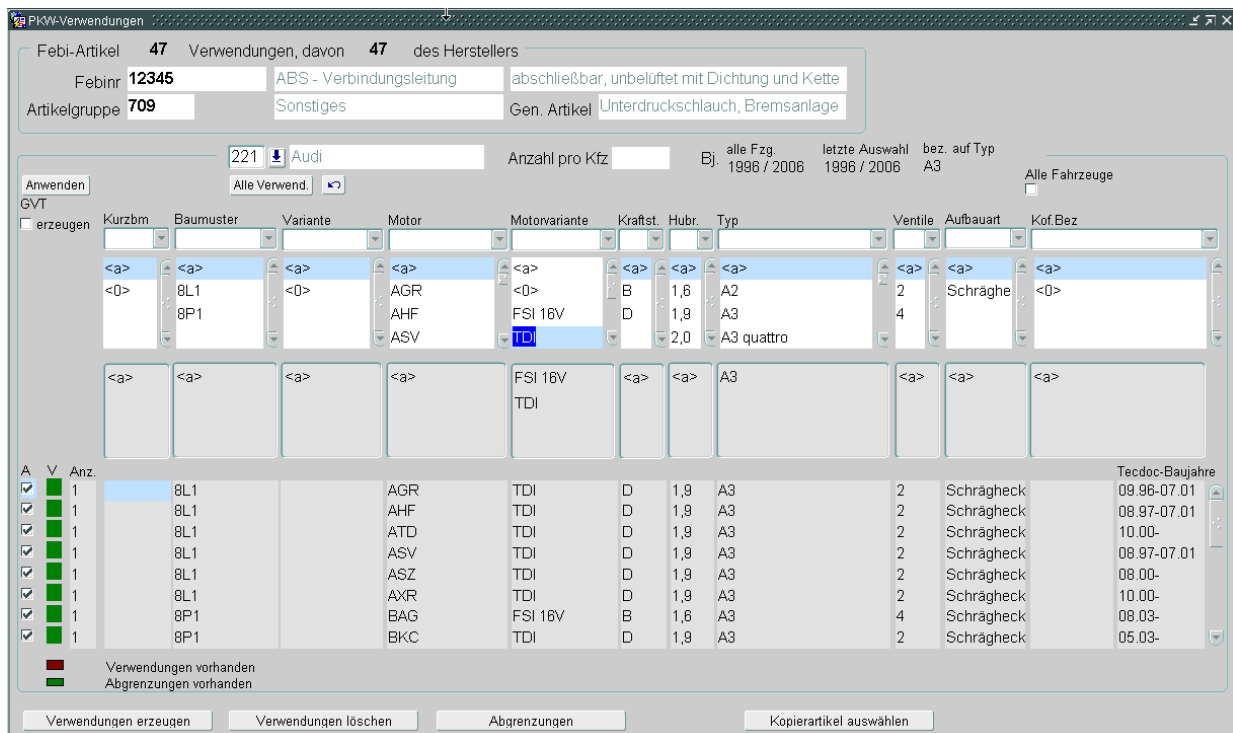


Abbildung 2: Pflege der Artikel-Verwendungen für PKW

Für die Anwendung sind ca. 250 Anwender registriert, wovon i. d. R. 70 bis 100 gleichzeitig angemeldet sind.

2. Die Technologie

In der aktuellen Ausbaustufe wird die Anwendung auf einer RAC-Datenbank, bestehend aus 3 Knoten, mit 10.2.0.3.0 Enterprise Edition betrieben. Das Middle tier bildet ein OAS 10.1.2.2.1 Enterprise Edition, aus dem die Forms- und Discoverer Services sowie mod_plsql verwendet werden.

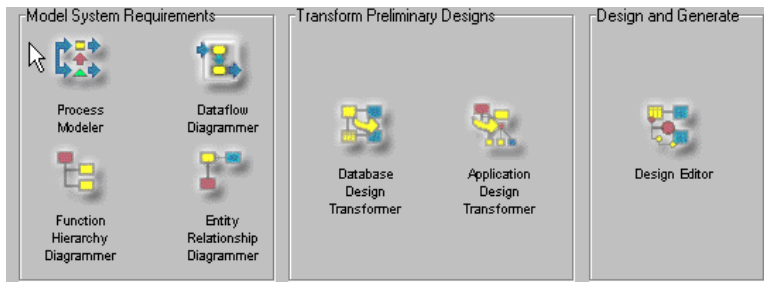
Das Datenbankschema beinhaltet ca. 660 Tabellen, 130 Packages/Procedures/Functions und 40 Trigger. Dem Benutzer stehen 35 Masken sehr unterschiedlichen Umfangs (100 KB bis 6 MB) zur Verfügung. Ein differenziertes Berechtigungssystem steuert die Bearbeitungsmöglichkeiten auf Form-, Block- und Item-Ebene und ermöglicht es dem Administrator, rollen- oder anwenderspezifisch Items ein- oder auszublenden. Mehrere Virtual Private Database-Policies gewährleisten, dass dem Anwender nur die seinem Zuständigkeitsbereich entsprechenden Informationen lesend bzw. verändernd zugänglich sind.

3. Die Entwicklungsmethode

Der Designer stellte von Beginn an das zentrale Entwicklungsinstrument dar. In früheren Projekten hatte ich mehrfach Gelegenheit, die vom Designer suggerierte Methode (Beschreibung der Anwendung mittels Funktionshierarchie, Prozess- und Datenflussmodell, Aufbau des Entitätenmodells, Überleitung dieser Modelle in physikalische DB-Strukturen und Generierung von Modulen) zu testen und zu evaluieren. Zusammenfassend ist festzustellen, dass der in der *Model System Requirements*-Phase, mit Ausnahme des ER-Diagrammers, gestiftete Nutzen



jedenfalls in kleineren und mittelgroßen Projekten den damit verbundenen Aufwand i. d. R. nicht rechtfertigt.



Im Projekt wurde entschieden, die Datenstrukturen nicht über das ER-Modell und den Database Design Transformer, sondern direkt im physikalischen Modell zu erstellen. Damit verbleibt der Design Editor als alleiniges Tool zur Modellierung der Anwendung.

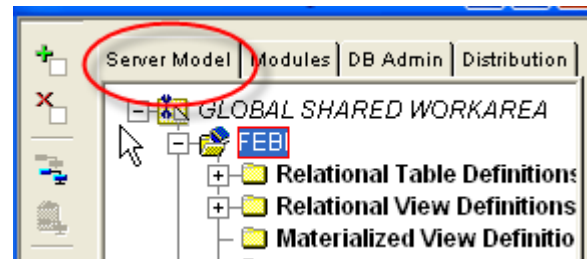
Abbildung 3: Ausschnitt aus dem Hauptmenü des Designers

Der Umgang mit dessen Hauptbestandteilen *Server Model* und *Modules* wird im Folgenden beschrieben.

3.1 Server Model Design

Ausnahmslos alle

- Tabellen
- Primary Key-, Foreign Key- und Check-Constraints
- Indizes
- Sequenzen



werden im Server Model angelegt.

Abbildung 4: Server Model Design

Es gilt die strikte Regel, dass es im produktiven Datenbankschema kein Objekt der genannten Typen gibt, das nicht im Repository repräsentiert ist. Alle DDL-Befehle werden aus dem Repository heraus generiert. Strukturänderungen erfolgen zunächst im Schema Model und werden je nach Umfang durch erneutes Generieren oder durch manuell erzeugtes DDL implementiert.

Für die Definition von Views im Designer stehen 2 Verfahren zur Wahl: Free Format und Wizard-gesteuert. Beiden gemein ist, dass sie gegenüber der Codierung im Editor kaum Erleichterungen bieten, insbesondere dann nicht, wenn komplexe Ausdrücke verwendet werden sollen. Zudem erlaubt der Wizard keine syntaktische Prüfung. Aus diesen Gründen ziehe ich es vor, View-Objekte manuell im Editor zu erstellen, in der Datenbank anzulegen und zu testen. Nach erfolgreichem Test wird das View-Objekt über Reverse Engineering in das Repository gebracht.

Auf diese Weise liefert das Server Model zu jeder Zeit eine vollständige und korrekte Dokumentation der produktiven Datenbank-Struktur zusätzlich der sich in Entwicklung befindlichen Objekte. Es bildet die Grundlage für die Abstimmung mit dem Kunden und für das Pflichtenheft.

3.2 Module Design

Zweifelloos liefert das Datenmodell die wichtigste Grundlage für die Generierung der Forms-Module. Weitere bedeutende Komponenten sind:

- die Object Library
- eine Template-Form
- die Präferenz-Hierarchie



Abbildung 7: Module Design

Im Projekt beinhaltet die Template-Form die Code-Teile und Objekte, welche die Berechtigungssteuerung erfordern, 2 PL/SQL-Libraries mit Standard-Funktionen und eine Reihe von Visual Attributes.

Der Zweck der Verwendung dieser Komponenten liegt darin, projektspezifische Standards für das Look & Feel und bestimmte Funktionalitäten zu definieren und auf diese Weise den Aufwand für nachträgliche Änderungen zu minimieren. Speziell das Setzen der Präferenzen ist zwar mit viel Testarbeit verbunden, und so mancher Weg führt in die Irre. Dennoch sollte der Aufwand nicht gescheut werden, liefert diese im Wesentlichen einmalig zu erbringende Arbeit doch einen bedeutenden Beitrag zum o. g. Zweck.

Trotz der grundsätzlich gegebenen Mächtigkeit des Forms-Generators gibt es meiner Meinung nach eine Reihe triftiger Gründe dafür, vom Anspruch abzurücken, eine 100%-Generierung zu erreichen:

- Einzelne Item-Typen (Hierarchical Tree, Bean Area, Chart Item) können nicht generiert werden. Manuelle Nachbearbeitung ist unvermeidlich, soll das Modul eines dieser Items enthalten
- Die Verteilung von Items eines Blocks auf verschiedene Canvases/Tabs war bis Version 9i nicht möglich.
- Der PL/SQL-Editor des Modul Designers ist nicht „state of the art“. Auch con Oracle gibt es hier gibt es mittlerweile wesentlich bessere Werkzeuge, die ein produktiveres Arbeiten ermöglichen.
- Ein funktionaler Test des mit dem PL/SQL-Editor entwickelten Codes im Designer ist nicht möglich. Daraus ergibt sich die Notwendigkeit, das Modul zu generieren und zu testen, den Code ggf. zu korrigieren (im Forms Builder, der Library bzw. der Datenbank) und ihn zurück in den Designer zu bringen, um das Repository konsistent zu halten. Mit solchen „round trips“ ist gewiss kein Produktivitätsgewinn zu erzielen.
- Schließlich kann es sehr mühsam – wenn nicht unmöglich - sein, ein gehobenen Ansprüchen genügendes Layout vollständig zu generieren.

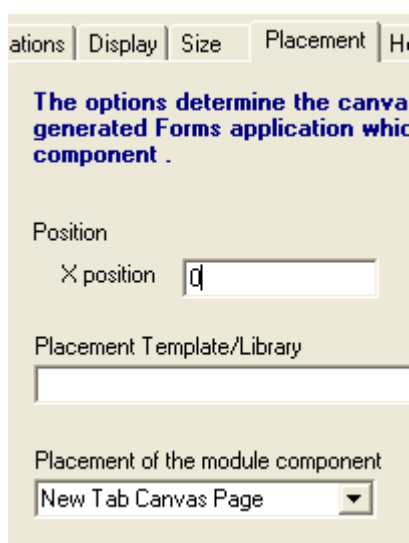
Neben den vorgenannten Gründen waren die Anforderungen des Kunden, in bestimmten Modulen von der Default-Forms-Funktionalität (z.B. Enter Query, Execute Query) abzuweichen sowie nicht generierbare Features in der Applikation zur Verfügung zu stellen, bestimmend für die Entscheidung, den 100%-Anspruch nicht zu verfolgen und statt dessen in einem pragmatischen Ansatz die Stärken des Forms-Generators zu nutzen.



Hier sind zu nennen:

- Entlastung von Standard-Codierungs-Aufgaben:
Validierungs-, Pre- und Post-Query-, Pre-Insert-Trigger, List of Values, Block-Synchronisation, auch bei Master-Master-Detail-Strukturen, Ein-/Ausblenden von Canvases
- Einheitliches Look & Feel wird sichergestellt
- Item-Eigenschaften werden vom Server Model bis in das Modul „durchgereicht“, somit entfällt die Definition immer gleicher Ausprägungen

Damit wird ein erheblicher Produktivitätsgewinn gegenüber der Modul-Entwicklung im Forms-Builder erreicht, auch wenn dessen Wizards genutzt werden. Diejenigen Modul-Anteile, die sich mit wenig Aufwand durch den Module Designer einstellen und spezifizieren lassen, werden selbstverständlich generiert.



Alle gebotenen Möglichkeiten werden genutzt, um dem gewünschten Ergebnis mit der Generierung so nahe wie möglich zu kommen. Die Eigenschaften von Windows, Blocks und Items werden gesetzt, soweit sie nicht aus dem Server Model vererbt wurden.

Mit Hilfe des Tap Stop Editors kann das Block-Layout relativ gut dem Zielformat angenähert werden, und die Bestimmung der Canvas-Typen und –Abmessungen legen das grundsätzliche Layout des Moduls fest.

Abbildung 8: Parameter für das Canvas Layout

Bei anderen Modulteilen dagegen, deren Hinterlegung im Repository einen zu hohen Aufwand verursachen würde oder die dort gar nicht abgelegt werden können, wird auf die vollständige Abbildung des Moduls im Repository zugunsten der besseren Effizienz verzichtet.

Gelegentliche „Kunstgriffe“ helfen dabei, den Anteil des generierten Codes am zu erstellenden Modul zu erhöhen: so ist es meiner Meinung nach zulässig, temporär (zur Generierungszeit) einen Foreign Key-Constraint zu deklarieren, um den Modul Generator zur Erzeugung einer Master-Detail-Beziehung zu bewegen, an einer Stelle, an der fachlich kein Foreign Key-Constraint gewünscht sein mag. Ebenso können temporäre Tabellen (zur Generierungszeit) als persistente deklariert werden, um darauf Foreign Key-Constraints anlegen zu können.

Funktionalität und/oder Layout des mit diesem Ansatz generierten Forms-Moduls entsprechen in aller Regel noch nicht vollständig den Anforderungen. Die Nachbearbeitung im Forms-Builder ist fast durchgängig erforderlich. In dieser Phase wird spezielle Geschäftslogik implementiert (PL/SQL in der Form, in Libraries, in der DB), die Maske wird um Items ergänzt, die im Repository nicht hinterlegt sind und das Layout erhält seinen endgültigen Schliff. Schließlich wird das Modul einem Funktions- und Integrationstest unterzogen, und es steht für die Auslieferung bereit.

Die Wartung und Weiterentwicklung der Module folgt demselben pragmatischen Ansatz: Idealerweise werden die im Server Model abgelegten Änderungen bzw. Ergänzungen der Datenstruktur in das betreffende Modul übernommen, ggf. mit Layoutinformationen angereichert,



und das Modul wird generiert. Im nächsten Schritt erfolgt die Einbettung des neuen Codes aus dem neuerlich generierten in das aktuelle Modul. Dieses Verfahren findet in ca. 80% der Fälle Anwendung. In Ausnahmefällen wird die gewünschte neue Funktionalität in einem isolierten Modul generiert, aus dem dann die neuen Bestandteile kopiert werden. In etwa 15% der Wartungsfälle handelt es sich um reine Änderungen der Geschäftslogik. Da diese ohnehin nicht im Repository hinterlegt ist, erfolgt die Änderung unabhängig vom Designer direkt im Forms-Modul.

4. Fazit

Dieser Ansatz stellt sicherlich nicht die „reine Lehre“ dar. Es handelt sich vielmehr um ein Vorgehen, das die im Designer angelegten Werkzeuge in pragmatischer Weise konsequent nutzt, um die ohnehin sehr gute Produktivität der Forms-Entwicklung (Forms-Builder) nochmals deutlich zu steigern. Dies ist das primäre Ziel des vorgestellten Ansatzes, dem andere, eher systematisch-methodische Aspekte untergeordnet sind.

Der Modul Generator liefert - ohne unangemessen großen Aufwand betreiben zu müssen - eine sehr gute, tragfähige Grundlage für die klassische Entwicklung. Das Repository wird, insbesondere was die Modul-Definitionen angeht, nur insoweit vervollständigt, als es Effizienzgewinne verspricht.

Im Projekt liegt der Leistungsanteil der Forms Generierung zwischen 60% und 90%, je nach Größe und Komplexität des Moduls sowie den Ansprüchen an Funktionalität und Layout. Die verbleibenden Anteile sind durch manuelle Entwicklung erbracht. Dadurch kann die Produktivität gegenüber der herkömmlichen Entwicklung im Forms-Builder um 40% bis 60% gesteigert werden.

Ein ähnliches Produktivitätsniveau zu erreichen muss die Stoßrichtung sein, in der ein Produkt entwickelt wird, das den Designer in den nächsten Jahren wird ersetzen müssen. In seinem Statement Of Direction zu den Oracle Tools¹ erklärt Oracle zwar einerseits, mit JDeveloper und ADF einen Ansatz zu bieten, der über eine visuelle und deklarative Entwicklungsumgebung die Produktivität und leichte Handhabbarkeit von Forms in die J2EE-Welt einführt, schränkt aber andererseits ein, nicht zu beabsichtigen, ein Werkzeug zur Migration der mit Forms, Reports und Designer erstellten Module bereitzustellen.

Das mit JHeadstart² verfolgte Ziel, die Modul-Metadaten des Repositories für ADF nutzbar zu machen, stellt einen Schritt in die richtige Richtung dar, reicht aber bei weitem nicht aus, dem oben erklärten Anspruch gerecht zu werden. Es liegt noch ein gutes Stück Weg vor Oracle, JDeveloper zu einer Rapid Application Development-Umgebung auszubauen, welche die Produktivität von Designer und Forms erreicht.

Viel Erfolg beim Einsatz von Trivadis-Know-how wünscht Ihnen

Rolf Wesp
Trivadis AG

Werdener Strasse 4
40227 Düsseldorf

Internet: www.trivadis.com

Tel: +49-211-58 66 64 70

Fax: +49-211-58 66 64 71

Mail: info@trivadis.com

Literatur und Links...

www.trivadis.com

¹ <http://www.oracle.com/technology/products/forms/pdf/10g/ToolsSOD.pdf>

² <http://download-uk.oracle.com/consulting/jhsdevguide1013.pdf>, Kapitel 13