ADF, Forms und .NET – alles vereint in einer Mobile-Scanner-App bei Volkswagen

Madi Serban, Bahar Us, Rastislav Misecka und Mathias Waedt, PITSS sowie David Christmann, Volkswagen AG

Das anspruchsvolle Projekt wurde im Jahr 2013 durchgeführt. Ziel war die Neuentwicklung einer alten Forms-2.0-Anwendung für Handscanner-Mobil-Geräte. Die neue Anwendung sollte auf unterstützten Plattformen laufen, toll aussehen, eine hervorragende Leistung anbieten und rechtzeitig entwickelt werden (zwei Monate Entwicklungszeit).

Dieser Artikel beschreibt die Überlegungen, die zur im Titel genannten Technologie-Auswahl geführt haben, Herausforderungen (wie Session Management, Tastatur-Bedienung, Sicherheit etc.), Architektur, das Zusammenspiel der Komponenten und die Lösungen, die es uns erlaubten, das Rennen gegen die Zeit zu gewinnen.

Alte Forms-Anwendungen für mobile Geräte

Oracle Forms ist in vielen Großunternehmen immer noch präsent. Man betreibt damit seit Jahrzehnten geschäftskritische Prozesse. Forms-Anwendungen funktionieren zuverlässig, mit exzellenter Leistungsfähigkeit und benötigen nur ab und zu Wartung oder Weiter-Entwicklung. Die meisten Forms-Anwendungen haben auch mindestens ein paar Masken, die für mobile Endgeräte konzipiert wurden. Diese sind in vielen Fällen von enormer Bedeutung, vielleicht sogar die meistbenutzten Masken, zum Beispiel für Handheld-Scanner in der Handels- oder Automobil-Industrie. Deshalb ist es sehr wichtig, dass diese mobilen Masken ständig funktionsfähig sind.

Was tun wir dann, wenn solche Masken Probleme bereiten? Zum Beispiel, wenn Oracle die alten Forms-Versionen nicht mehr unterstützt und diese modernisiert werden müssen. Die einfachste und stressfreieste Lösung wäre eine Migration nach Forms 11g. Diese Lösung ist aber aufgrund spezifischer Hardware- und Software-Konfigurationen oder Benutzer-Anforderungen nicht für alle Situationen passend.

Obwohl die nach Forms 11g migrierten Masken Web-fähig sind und ganz gut aussehen, sind sie trotzdem auf manchen mobilen Geräten nicht mehr lauffähig, weil die Ressourcen für ein Forms Applet schlicht und einfach fehlen. Die Remote-Desktop-Lösung ist manchmal hilfreich, aber nur mit großen Security- und Konfigurations-Problemen erreichbar. Die Remote-Lösung (Citrix) war in diesem Fall auch nicht kompatibel mit den verfügbaren Geräten.

Die Optionen für Upgrade, Weiterentwicklung, Migration und Neuentwicklung müssen dann gut gewichtet sein, um eine optimale Entscheidung zu finden und die Anwendungslebensdauer um weitere Jahrzehnte zu verlängern. Dafür nahmen die Autoren zuerst die Anforderungen unter die Lupe.

Die neuen Ziele für mobile Datenbank-Anwendungen

Natürlich soll die neue Anwendung auch weiterhin die alten Funktionalitäten genauso gut abdecken wie die alte:

- Endkunden erwarten eine mindestens gleichbleibend tolle Leistung, und das ist nicht bei allen Technologien so selbstverständlich wie bei Oracle Forms.
- Hot-Key-Funktionen sind in Oracle Forms sehr beliebt. Leider sind sie in Web- und Touchpad-Anwendungen nicht üblich und benötigen einen sehr hohen Programmierungsaufwand. In Browser-Anwendungen bedeutet das meistens viel JavaScript und in Desktop-Anwendungen das Überschreiben der normalen Komponenten-Funktionalität. Dies ist nicht zu empfehlen, aber leider auch in vielen Situationen nicht zu vermeiden.

Hinzu kommen meistens folgende Neu-Anforderungen:

- Unterstützung für neue Geräte und neue Betriebssysteme wie Android, iOS, etc., aber auch für alte Geräte. In dieser Situation war die Unterstützung für Microsoft Windows CE 5 nicht zu umgehen.
- Strategische Richtung für andere Programmiersprachen wie Java
- Zertifizierte Laufzeitumgebung
- Möglichst die Verfolgung Oracle-strategischer Empfehlungen und Best-Practices
- Sicherung der Investition, um die zukünftige Entwicklung und den Migrationsbedarf zu verringern
- Gutaussehende Benutzeroberfläche – schließlich sind Endkunden heutzutage modernste Apps für Mobil-Telefone gewöhnt und wünschen sich natürlich, eine entsprechende Benutzeroberfläche auch im Arbeitsumfeld zu haben
- Kiosk/App Modus – also Anwendung nicht einfach im Browser öffnen, sondern als App
- Funktionelle Erweiterungen

Und das alles soll mit minimalem Zeitaufwand und maximaler Qualität erfüllt werden. Für dieses Projekt-Beispiel ging es um zwei Monate Entwicklungszeit. Es war ein Rennen gegen die Zeit und das hat motiviert, ständig nach klugen Lösungen zu suchen. Es war sicherlich eine spannende Research- und Entwicklungs-Zeit. Warum Research? Zwei Wochen waren allein notwendig, um die optimale Technologie-Auswahl und Architektur zu definieren.

Die Technologie

Wenn man nach Alternativen für Forms sucht, dann richten sich die ersten Gedanken meistens auf ADF und Apex. Danach zieht man auch Java-Open-Source und Microsoft .NET in Erwägung. ADF und Apex bieten exzellente Lösungen für mobile Oracle-Datenbank-Anwendungen wegen der hohen Kompatibilität mit den restlichen Oracle-Anwendungen und auch der Möglichkeiten für eine vereinfachte Migration. Die ADF-Optionen für eine mobile Anwendungsentwicklung waren:

- Oracle ADF Faces Rich Client Components
- Oracle ADF Mobile Browser

Leider konnte keine dieser Optionen alleine alle Anforderungen erfüllen. Die Zertifizierungsmatrix für Oracle ADF Mobile Browser sieht beim ersten Blick in Ordnung aus (siehe Tabelle 1). Bei näherer Betrachtung bestätigte sich jedoch die alte Regel: Der Teufel steckt im Detail. Eignungstests haben schnell gezeigt, dass eine ADF-Anwendung in der mobilen Version des IE für Windows CE 5 zwar prinzipiell lauffähig war, jedoch nicht ganz in gewünschtem Umfang. Das Problem ist der für Embedded Systems angepasste Internet Explorer Browser Version 5 oder 6, der dafür nicht ausreichend JavaScript- und PPR-fähig (partial page rendering, alias Ajax) ist. JavaScript und PPR waren für die Hot-Key-Funktionen jedoch ein absolutes Muss.

Die Lösung

Wie eingangs im Titel erwähnt, bestand die Lösung aus einer Kombination aus Oracle-Datenbank, ADF-Anwendung und .NET-Benutzeroberfläche (siehe Abbildung 1). Die Forms-Funktionalität wurde dafür in drei Schichten migriert: eine Thin-.NET-Benutzeroberfläche sowie ein Thick-Datenbank- und ein ADF-Layer, alles schnell in fünf Phasen durchgeführt (siehe Abbildung 2):

- Phase 1: Forms-Vorbereitung, Migration PL/SQL Business Logic in die Datenbank
- Phase 2: Upgrade Oracle Forms 3 nach 11g
- Phase 3: Migration Oracle Forms nach ADF
- Phase 4: Entwicklung der .NET-Benutzeroberfläche und Integration mit ADF
- Phase 5: Test und Go-Live

BROWSER	ADF Mobile
BlackBerry Browser 4	Certified
WebKit-based mobile browsers (iPhone Safari, Android Chrome, Nokia S60)	Certified
Access NetFront 3	Certified
OpenWave (UP Browser) 7	Certified
Opera Mini 8	Certified
Pocket Internet Explorer for Windows Mobile 5, 6	Certified
Mobile Internet Browser 2.0 (Motorola)	Certified
Other Basic XHTML mobile browsers	Supported

Tabelle 1: Mobile Browsers zertifiziert für Oracle ADF Mobile Browser siehe http://www.oracle.com/technetwork/developer-tools/jdev/jdev11gr2-cert-405181.html

Die größten Herausforderungen

Wie migriert man 90 Prozent der Forms-PL/SQL-Business-Logik in die Datenbank, sodass diese optimal von ADF konsumiert werden kann? Was muss man hier berücksichtigen? Nachdem die Lösung für alle Phasen festgelegt war, konnten die Bausteine nacheinander entwickelt und integriert werden. Es war klar, dass der Prozessfluss als Services gebaut werden sollte, um den unabhängigen Informationsaustausch zwischen der Benutzeroberfläche und dem Backend zu erlauben und so die Datenkonsistenz zu erhalten.

Um diese Services herzustellen, wurde die Forms-Business-Logik (Trigger und Program-Units, die den Prozessfluss abbildeten) in die Datenbank migriert. Dafür hat man die Benutzer-Interaktion, insbesondere die notwendigen Hot-Key-Funktionen, als Ansatzpunkt gewählt und mithilfe der PITSS.CON-Top-Down- und Bottom-Up-Analyse die gesamte Kette von Abhängigkeiten und relevanter Geschäftslogik identifiziert und in Datenbank-Pakete migriert. PITSS.CON-BL-Assistant hat sich um die Migration gekümmert und durch die automatische Erkennung der Forms-Bind-Variablen und die Umwandlung in Parameter viel Projektzeit gespart. Diese Parametrisierung hatte auch eine schöne Nebenwirkung: Viele Prozeduren oder Funktionen, die in den Forms-Modulen vervielfacht waren, konnten durch die Parametrisierung eliminiert werden, was langfristig zu vereinfachter Wartung führt.

Als die Technologien in der Datenbank, in den Web-Services und auf der Benutzeroberfläche unabhängig voneinander funktionierten, waren die absolut getrennten, atomaren Funktionen notwendig. Zum Beispiel wurden in den PL/SQL-Paketen keine globalen Variablen benutzt, um eine geteilte Datenbank-Session zu erlauben. Weil manche PL/SQL-Datentypen kein Interface im JDBC-API haben, wurden stattdessen einfache Datentypen in den PL/SQL-Interfaces durch Web-Services ersetzt. Was man hier noch ganz am Anfang berücksichtigen musste, war die Portierung zentraler Funktionalitäten wie Fehlerbehandlung und Meldungen von Forms zur Datenbank.

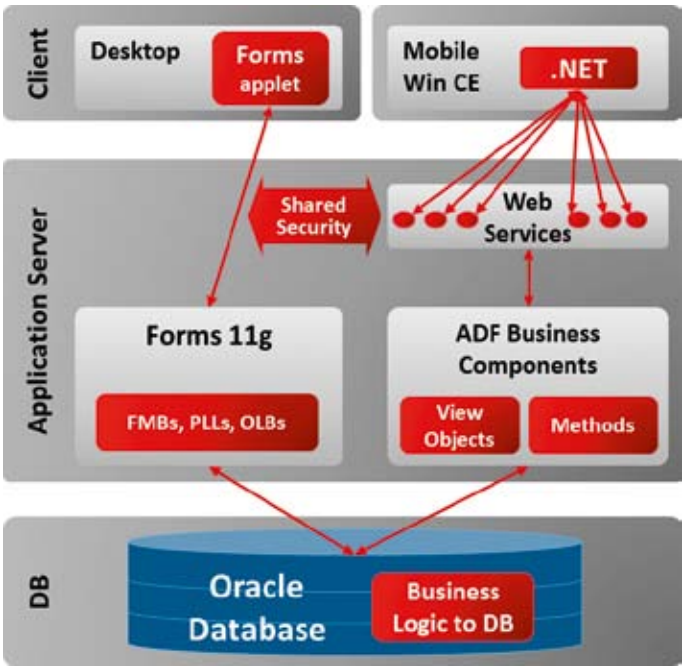

Abbildung 1: Forms, ADF-Server und .NET-Architektur

Solche Funktionalitäten waren in allen Forms-Triggern, Prozeduren und Funktionen präsent und ohne eine Datenbank-zentralisierte Fehlerbehandlung wäre es nicht möglich, die Logik richtig zu migrieren.

Zusätzlich wurde die Logik in der Datenbank in bestimmten Situationen weiter aufgeteilt oder parametrisiert, um den Prozess gegebenenfalls zu unterbrechen und um den Informationsaustausch mit der Benutzeroberfläche zu ermöglichen. Die Datenbank-Schnittstellen hat man mit Parametern ausgebaut, die entweder in der Datenbank-Logik benötigt oder in der Benutzeroberfläche abgebildet wurden.

Die ganze Migration der Logik war eine Herausforderung, auch aufgrund des Debuggens und Testens in unterschiedlichen Programmiersprachen und Entwicklungsumgebungen. Aber es hat sich gelohnt: Die Logik liegt nun sicher in der Datenbank und kann von dort sowohl von Forms als auch von ADF, .NET oder in Zukunft auch von anderen Technologien konsumiert werden.

In der Phase 2 erfolgte das Upgrade von Oracle Forms 3 nach 11g. Die Forms-Module sind durch die vielen Entwicklungsjahre sehr komplex geworden. Wegen dieser Komplexität und auch wegen der teilweise veralteten Dokumentation ist die Weiterentwicklung sehr schwierig geworden. Deshalb wurde hier PITSS.CON eingesetzt, um den Großteil der Anwendung nach Forms 11g zu migrieren. Dies war die einzige passende Lösung und erlaubte innerhalb kürzester Zeit die Umstellung auf eine moderne Technologie.

Eine leistungsfähige ADF-Web-Services-Architektur

In der Phase 3 ging es um die Fragen „Wie baut man eine leistungsfähige ADF-Web-Services-Architektur?“ und „Welche Herausforderungen sind zu bewältigen?“ Die Middle-Tier dieses Systems wurde auch so schlank wie möglich gehalten. Dadurch ist die Wartbarkeit der Kern-Funktionalität in der Datenbank zentralisiert und von den übrigen Teilsystemen weitestgehend entkoppelt. Da das System auch in Umgebungen mit eingeschränkter Netzwerkverbindung eingesetzt werden muss, hat man die Kommunikation zwischen Handheld und Server auf ein Mindestmaß gesenkt.

Eine der Eigenschaften von Forms ist die Möglichkeit, UI und Business Logic (BL) ganz einfach zusammenzulegen. Was in Forms effektiv sein kann, erweist sich in Migrationen auf Model-View-Controller-Frameworks (MVC) üblicherweise als trickreicher Spießrutenlauf. Genau dies war eine der weiteren Herausforderungen an das neue verteilte System. Die Hauptaufgabe war hier die Entflechtung von UI und BL, um möglichst viel BL in die Datenbank zu verlagern (Wartbarkeit). Typisch sind hier Prozessstränge, die, durchsetzt mit Rückfragen an den Benutzer, unterschiedliche Verläufe annehmen können. Es gilt auch hier, den besten Mittelweg unter Reduzierung des Communication-Payload und der Anzahl der Request-Response-Zyklen (Server-Roundtrips) zu finden sowie dabei die vorhandene Funktionalität zu gewährleisten und sogar zu verbessern. Im Web-Services-Bereich wurden größtenteils Standard-Komponenten von Oracle ADF eingesetzt.

Ein weiterer Punkt war der großzügige Einsatz von Datenbank-Prozedur-Aufrufen. Wo Forms üblicherweise mit dedizierter Datenbank-Verbindung arbeitet, wird im ADF-Bereich Connection Pooling eingesetzt, das Connection Management also extern (etwa vom WebLogic Server) verwaltet. Der Vorteil hierbei ist, dass der Server selbst

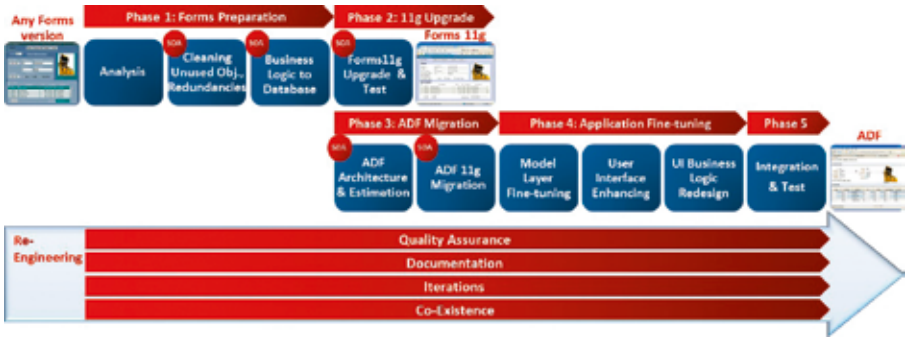

Abbildung 2: Forms nach 11g und ADF Modernisierungsprozess mit PITSS.CON

die Kontrolle über diese Ressourcen hat und seine Leistungsfähigkeit diesbezüglich dynamisch steuern kann. Die Herausforderung hier war, trotz wechselnder Datenbank-Connections und Sessions einen konsistenten Datenstand zu bewahren.

Zusätzlich dazu war es Bedingung, dass auf dem WebLogic Server, abhängig von Handheld und Benutzer (also eingehenden Web-Service-Requests), unterschiedliche Datenbanken angesprochen werden konnten. Das heißt, während der Laufzeit musste dynamisch die richtige Connection-Art aus dem Pool herausgefischt werden. Eine weitere wichtige Anforderung war, das System ohne große Änderungen an mehreren Standorten einsetzen zu können. Dabei sollten identische Datenstrukturen, aber mit unterschiedlichem Datenstand, angesprochen werden. Mehrsprachigkeit war ebenfalls eine Bedingung. Meldungen an den Benutzer müssen in der für ihn eingestellten Sprache erfolgen. Selbstverständlich muss dies alles unter den Gesichtspunkten einer gesicherten Kommunikation erfolgen.

Abbildung 3: Die neue Benutzeroberfläche

Alles außer Routine

In Phase 4 erfolgten in sechs Wochen die Entwicklung der .NET-Benutzeroberfläche sowie die Integration von .NET und ADF. Die Anwendung für die Handhelds soll an mehreren Standorten zum Einsatz kommen. Aus einer solchen Konstellation heraus ergibt sich fast unweigerlich, dass auch unterschiedliche Geräte, zum Teil auch von mehreren Herstellern, mit unterschiedlichen Eigenschaften und Fähigkeiten in Gebrauch sind. Für die Entwicklung einer solchen Anwendung hat dies zur Folge, dass man sich praktischerweise für eine Implementierungs-Plattform entscheiden muss, die von allen eingesetzten Geräten unterstützt wird. In dem vorliegenden Fall hat sich für die gestellten Anforderungen das „.NET Compact Framework 2.0 SP2“ als die geeignetste Plattform herauskristallisiert.

Der Einsatz von .NET bietet viele Vorteile, nicht zuletzt auch den Umstand, dass die für das Compact Framework kompilierten Anwendungen in den meisten Fällen auch unter der Desktop-Version des Frameworks lauffähig sind. Das hat erhebliche Vorteile beim Support und der Wartung: Man kann die Anwendung sehr gut auf einem ganz normalen PC betreuen, ohne die Notwendigkeit, immer ein Handheld bereitzuhalten.

Doch es gibt auch eine Schattenseite: Das Compact Framework hat gegenüber der großen Desktop-Version einen etwas reduzierten Funktionsumfang. Damit nicht genug – man muss auch feststellen, dass Microsoft bei der Wahl der wegzulassenden Features nicht immer ein glückliches Händchen hatte. So fehlen einige Funktionalitäten, die schmerzlich vermisst werden und die man sich dann selbst nachprogrammieren muss.

Oberflächen-Gestaltung

Die Trends im „Consumer Electronics“-Bereich (Smartphones) sind klar: größer, schärfer, brillanter. Immer höhere Auflösung – HD Displays gehören ja schon fast zum Standard, es gibt sogar schon einige Exemplare mit einer Full-HD-Auflösung. Bei den mobilen Geräten für die Industrie sieht es noch anders aus. Hier zählen völlig anderen Werte: Stoßfestigkeit, Kältebeständigkeit, Kratz- und Staub-Unempfindlichkeit, Bedienbarkeit mit Handschuhen oder Integration mit Bar-Code-Scannern.

Aber nicht selten sind auch Geräte mit Display-Auflösungen von nur 240x300 Pixeln in Gebrauch. Insbesondere der letztgenannte Punkt kann einen Full-HD-Desktop- verwöhnten Entwickler vor unerwartete Herausforderungen stellen. Schon allein die bündige Positionierung der Controls im visuellen Designer für diese Auflösung wird zur gänzlich neuen Erfahrung und erfordert eine sichere Hand – und die Geduld eines Uhrmachers.

Überhaupt ist die Aufgabe, das richtige Layout für die einzelnen Masken zu finden, keine einfache. Beim Entwurf muss man gleich mehrere, zum Teil widersprüchliche Aspekte berücksichtigen. In erster Linie sind das natürlich die Hardware-Möglichkeiten der eingesetzten Geräte. Nicht alle Geräte verfügen über einen Touchscreen. Und selbst wenn sie es täten, entspräche eine rein Touch-zentrische Bedienung nicht den Gegebenheiten des Einsatzorts. Ein Touchscreen mit einer Auflösung von nur 240x300 Pixeln im Winter auf einem kalten Parkplatz mit dicken Handschuhen treffsicher bedienen zu müssen – keine allzu verlockende Vorstellung.

Andererseits sind Smartphones und andere mobile Geräte (Tablets) heutzutage so verbreitet, dass sie zu einer Art Etalon des GUI-Designs geworden sind. Daher muss man sich wirklich anstrengen, wenn die eigene Oberfläche im Vergleich dazu nicht als altbacken aussehend wahrgenommen werden soll. Selbst wenn es sich nur um eine Industrie-Anwendung handelt.

Dieser Einfluss fand seinen stärksten Ausdruck in der Gestaltung des Anwendungs-Menüs. Große Icons, die man gegebenenfalls auch noch mit Handschuhen gut treffen kann und das Scrollen mittels Wischbewe-

gungen (falls vom Gerät unterstützt) machen das Auffinden der gewünschten Funktionalität zum Kinderspiel (die Bedienung mit den Pfeiltasten der Tastatur ist aber natürlich auch möglich). Bei der Gestaltung der restlichen Masken überwogen die funktionalen Anforderungen und das Bestreben, bei der – relativ gesehen – geringen Auflösung so viele relevante Informationen wie möglich darzustellen. Einer der Tricks, um das letztgenannte Vorhaben zu verwirklichen, war beispielsweise die farbliche Hervorhebung (eine Art „dritte Dimension“) der Datensätze mit speziellen Merkmalen (etwa Stornierungen) anstatt dedizierter Felder (siehe Abbildung 3).

Wichtig bei der Entscheidung war auch die Tatsache, dass das Haupt-Eingabegerät in den meisten Fällen der Barcode-Scanner ist. Wo sich die direkte Tastatur-Eingabe nicht vermeiden ließ, wird dem Anwender mit allerlei Komfort-Features unter die Arme gegriffen – etwa mit einem automatischen Sprung zum nächsten Eingabefeld, nachdem die Eingabe im aktuellen Feld als vollständig erkannt wurde. Die restliche Bedienung mittels Hotkeys orientierte sich an den Anforderungen und dem Umstand, dass es sich hierbei um die Migration einer bestehenden Anwendung handelt. Die Beibehaltung der gewohnten Abläufe trägt entscheidend zur schnellen Akzeptanz der Neuerung bei den Endanwendern bei.

Ein weiterer Bereich, der sich stark an modernen mobilen GUIs orientiert, ist die Navigation zwischen den einzelnen Seiten. Es wurde (mit Ausnahme von Fehlermeldungen und wichtigen Benachrichtigungen) weitestgehend auf Pop-up-Fenster verzichtet und stattdessen auf das sogenannte „Browser-Konzept“ gesetzt – für eine zusätzliche Information oder durchzuführende Aktion (Beispiel: LOV-Auswahl) wird eine separate Seite angezeigt, die die gesamte Bildschirmfläche einnimmt. Die alte Seite und deren Inhalt werden jedoch nicht verworfen, sondern beibehalten und nach Abschluss der Aktion wieder angezeigt. Nicht unähnlich einer Internet-Browser-Navigation, daher auch der Name. Dadurch wird verhindert, dass die wertvolle Anzeigefläche durch „Fenster-Chrome“ überproportional vereinnahmt wird.

Besondere Aufmerksamkeit erforderte auch das Thema „Mehrsprachigkeit“. Hier hat sich zum einen der reduzierte Umfang des Compact Frameworks bei der Steuerung der Ländereinstellungen des Benutzers bemerkbar gemacht: In der Desktop-Version des Frameworks ist es vergleichsweise einfach, die Sprache des UI durch Setzen der Eigenschaft „CurrentUICulture“ zu Laufzeit zu ändern. Leider wurde diese Eigenschaft in der Compact-Version des Frameworks eingespart und das UI läuft immer mit den Spracheinstellungen des Betriebssystems. Dies ist allerdings nicht immer ausreichend. Manchmal ist es einfach wünschenswert, die Oberflächensprache den (oft in der Datenbank gespeicherten) Einstellungen des aktuell angemeldeten Benutzers dynamisch anzupassen. So war das auch im vorliegenden Fall.

Eine weitere Komplikation bestand in der Tatsache, dass die ursprüngliche Forms-Anwendung ihre lokalisierten Texte in der Datenbank speicherte. Immerhin ist es einfacher, die Überschriften für eine neue Sprache in eine Datenbank-Tabelle einzupflegen (oder die für eine bereits existierende zu bearbeiten), als eine neue Dialogvorlage in Visual Studio zu bearbeiten und die Anwendung neu kompilieren zu müssen. Doch ganz so einfach ist es nicht: Einige Frameworks – und darunter auch das eingesetzte Windows Forms – betrachten die Lokalisierung in etwas breiterem Kontext und beziehen zusätzlich zum Text auch einige weitere Eigenschaften mit ein – etwa die Positions- und Größen-Angaben der einzelnen Controls.

Wenn man bei der Suche nach einer Lösung nicht auf Dienste des visuellen WinForms Designer in Visual Studio verzichten möchte, wird die Lösung des Problems noch zusätzlich durch die Tatsache erschwert, dass eben dieser Designer im Endeffekt als Code-Generator arbeitet, mit eher wenigen Möglichkeiten für den Entwickler, sich in den Prozess einzuklinken. Aber mit ein paar Tricks wurde auch diese Hürde gemeistert und alle beschriebenen Plattform-Verschiedenheiten zu einem einheitlichen Ganzen verschmolzen.

Test und Go-Live

Nach der Benutzeroberflächen-Gestaltung und -Logik kommt die Test-Phase: Die Anwender konnten in Ingolstadt und Wolfsburg bereits die mobile Masken produktiv nutzen und sind sehr zufrieden: Die Applikation funktioniert sehr gut und sieht hübsch aus. Die Performance ist auch in Ordnung – kein sichtbarer Unterschied zum früheren Forms-System. Und weil die Benutzeroberfläche nicht strukturell geändert, sondern modernisiert und teilweise vereinfacht wurde, konnten die Anwender sich gleich im neuen System zurechtfinden.

Es war wirklich ein sehr sportliches Projekt und es hat sich gelohnt. Die Anwendung wurde schnell geliefert und man hat erreicht, was man erreichen wollte: Dass die Endbenutzer die gleichen Funktionalitäten auch mit moderneren Mobile-Geräten bedienen können und die gesamte Oracle-Anwendung nun ein neues Leben hat.

Madi Serban, Mathias Waedt,
Rastislav Misecka, Bahar Us
mserban@pitss.de

David Christmann
david.christmann@volkswagen.de