Modernisierung einer Software AG EntireX basierten Lösung durch Oracle Fusion Middleware

Markus Zehnder, Trivadis AG, Bern

Schlüsselworte:

Case Study Middleware Modernisierung, Oracle Service Bus, Validierungskonzepte

Einleitung

Diese Case Study schildert die bis anhin gemachten Projekterfahrungen in der Modernisierung einer grösseren Software AG EntireX basierten Middleware Lösung durch die Oracle Fusion Middleware 11g.
Die NATURAL Services und ADABAS Datenbanken werden durch die Oracle SOA Suite und Oracle Service Bus (OSB), Java EE und Oracle Enterprise Datenbank 11gR2 ersetzt. Dabei haben die Client Applikationen weiterhin die Möglichkeit sich mit der bestehenden EntireX Schnittstelle anzubinden.
Der Fokus dieser Case Study liegt auf den Architektur Entscheidungen und der Herausforderung die alten Anwendungen möglichst transparent ohne Änderungen der Schnittstellen anzubinden und gleichzeitig das neue System nicht mit den alten Konzepten zu beeinträchtigen.
Weiterhin werden unsere Lösungen bezüglich Schattenbetrieb während der Inbetriebnahme und die Möglichkeiten der fachlichen Datenvalidierung mittels Oracle Business Rules und Schematron aufgezeigt.

Ausgangslage

Mit dem Projekt *Daten-Management* 2010 (DM2010) soll im Auftrag vom Schweizer Bundesamt für Strassen ASTRA das *Informatiksystem Verkehrszulassung* (IVZ) gebaut werden. Dieses löst das bestehende zentrale System - umfassend aus Motorfahrzeuginformationssystem, Fahrberechtigungs- und Administrativmassnahmen-Register (MOFAD) - ab.
Trivadis arbeitete bereits in der Voranalyse- und Konzeptphasen mit und erhielt den WTO Zuschlag für den Systemdesign und Detailspezifikation mit Option zur Realisierung. Bevor im Dezember 2011 vom Projektausschuss der Entscheid zur Realisierung gefällt wird, muss mit 2 Prototypen (PT) bewiesen werden, dass

- die vorgeschlagene Systemlösung technisch machbar ist (Anbindung der EntireX Client Schnittstellen und Realisierung eines Geschäftsprozesses, PT1), und
- die Daten richtig migriert und mindestens die bisherige Performance erreicht wird (PT2).

Der erste Prototyp wurde im Sommer erfolgreich abgeschlossen und die Ergebnis-Analyse von PT2 hat im September 2011 begonnen.

Herausforderungen

Viele Altsysteme sind von monolithischer Natur, kommunizieren über proprietäre Schnittstellen, sind nicht in heute gängigen Sprachen implementiert und laufen auf Mainframes. Fehlende oder unvollständige Dokumentationen machen die Aufgabe noch ein Stück interessanter. Dies sind alles bekannte Herausforderungen in Integrationsprojekten. Aus diesem technischen Problembereich wird nachfolgend die Schnittstellenproblematik im DM2010 Projekt näher betrachtet. Der Fokus liegt auf der Integration mittels OSB und dem Zusammenspiel mit der SOA Suite.

EntireX Client Anbindung
Das grösste Risiko bestand in der Integration der proprietären EntireX Kommunikation. Im Altsystem werden den Clients sogenannte „EntireX Konnektoren“ zur Verfügung gestellt, welche die Kommunikation abstrahieren und mit einem fachlichen API in Java, .NET und C versehen. Eine Projekt Anforderung besagte, dass die Client-Schnittstellen nicht geändert werden dürfen und ein transparenter Wechsel vom alten zum neuen System stattfinden muss. Eine Möglichkeit die Konnektoren auszutauschen bestand nicht. Der Hauptfokus im Prototyp 1 lag auf der Evaluierung von verschiedenen Integrations-Varianten, um die bestmögliche Lösung im Hinblick auf Performance, Integrationsaufwand und Realisierbarkeit zu bestimmen.
Folgende Varianten wurden evaluiert:

- Einsatz der Software AG EntireX XML/SOAP RPC Server und Web Services Stack (WSS):
  Der SOAP RPC Server ermöglicht den EntireX Clients Web Services aufzurufen. Der WSS dient zum Bereitstellen von Web Services von bestehenden EntireX Services.
- EntireX Java & Enterprise JavaBeans Wrapper:
  Aus den EntireX Schnittstellen-Definitionen (IDL) lassen sich mit der Entwicklungsumgebung Client- und Server-Wrapper für verschiedene Sprachen generieren.
- OSB Custom Transport:
  Entwicklung von Transport Providern mit Hilfe der Java Client- und Server-Wrapper.

Die Evaluierung zeigte, dass für unseren Anwendungsfall die erste Variante mit den Software AG Komponenten unter Einbezug sämtlicher Anforderungen die beste Wahl ist. Ausschlaggebend waren folgende Punkte:

- Die Clients verzichten auf Transaktionen (EntireX Conversations) und es sind nur atomare Operationen vorhanden. Ohne diesen glücklichen Umstand wäre die direkte Integration mit den EntireX XML Komponenten nicht möglich gewesen oder bedingte grössere Änderungen am bestehenden System.
- Die Software AG XML Komponenten haben sich in den Prototyp Phasen bewährt und es traten nur minimale Probleme auf. Die Performance-Einbussen waren zudem sehr gering.
- Die automatische Generierung der XML Mapping Definitionen aus den EntireX IDL Dateien verursacht den kleinsten Integrationsaufwand. Zudem kann mit der SOAP Schnittstelle im OSB ein sauberes Versionierungskonzept implementiert werden, d.h. es können auf einfache Weise während dem Betrieb mehrere Versionen aktiv sein. Zudem ermöglicht dies sehr schnelle Anpassung an Schnittstellenänderungen während der Realisierungsphase. Im Idealfall beschränken sich die Änderungen auf Datenmappings im OSB und nicht auf zusätzliche Programm Anpassungen wie mit den anderen Varianten.
- Architektonisch besteht eine saubere Trennung zwischen den EntireX Komponenten und dem IVZ System. Die Kommunikation geschieht ausschliesslich mittels Standard SOAP Meldungen. Dies erlaubt zusätzliche Integrationstests auf Ebene Web Services mit bewährten Tools wie soapUI.

Auch aus betrieblicher Sicht ergeben sich durch diese Lösung signifikante Vorteile:

- Zusätzliche Flexibilität bezüglich Netzwerkzonen-Übergängen beim Betreiber – das IVZ System wird in einer anderen Zone betrieben. Es sind keine Sonderbewilligungen erforderlich, da die bestehenden Web Service Gateways benutzt werden können.
- Die Software AG XML Komponenten werden bereits im bestehenden System eingesetzt und beim Betreiber existiert vorhandenes Knowhow für den Betrieb.

CRUD Schnittstelle
Für alle fachlichen Entitäten besteht im Altsystem eine CRUD Schnittstelle und es werden grundsätzlich immer alle Daten gelesen wie geschrieben. Die Daten Strukturen entsprechen dabei fast

nahtlos den de-normalisierten Datenbank Tabellen. Daraus werden auch immer zusätzliche Daten mitgeliefert, ob sie gebraucht werden oder nicht. Darunter fallen z.B. bei einer Person alle Adress-, Zahlungsinformations- und medizinische Nachkontroll-Termine.
Die Herausforderung für IVZ bestand darin, die Schnittstellen zu übernehmen aber die Datenhaltung in einem relationalen Modell zu modernisieren um alte Limitationen aufzuheben. Dazu gehören auch die statischen Datenstrukturen in 1:n Beziehungen überzuführen. Die Separierung wurde mit dem OSB gelöst, um das restliche System nicht zu beeinträchtigen:

- Die alten Schnittstellen und deren Datenstrukturen sind durch das XML Mapping (siehe vorheriges Kapitel) auf dem OSB realisiert.
- Die Daten werden im OSB ins kanonische Datenformat transformiert.
- Mittels Service Orchestrierung und dem Einsatz von zusätzlichen Fassaden auf den Daten Services werden die alten Schnittstellenoperationen abgebildet. Dadurch sind die IVZ Daten Services auf das neue Datenmodell ausgelegt und nicht ein Abbild der alten Schnittstellen.

Warenkorb Prinzip als generische Business Use Case Schnittstelle
Für Geschäftsfälle mit verschiedenen Entitäten existieren im Altsystem keine klassischen Business Services! Stattdessen existiert ein Warenkorb Prinzip für die ungefähr 30 Geschäftsfälle im Motorfahrzeuginformationssystem bei denen jeweils ein Halter, ein Nummernschild und bis zu 90 Fahrzeuge involviert sind. Aktionen auf nur eine Entität, wie z.B. eine Adressänderung, sind nicht als Geschäftsfall realisiert und müssen mit den CRUD Schnittstellen durchgeführt werden. Das Interessante daran ist, dass beim Einbuchen des Warenkorbes der auszuführende Geschäftsfall nicht angegeben werden kann, sondern als Antwort vom Server zurückgeliefert wird! D.h. die Clients müssen alle Daten in den Warenkorb laden, wie sie nach dem Geschäftsfall auszusehen haben und die Aufgabe des Servers ist herauszufinden welcher Geschäftsfall auszuführen ist. Die Logik der Datenänderungen liegt somit beim Client. Die Aufgabe des Servers beschränkt sich auf die Datenvalidierung, eventuelle Meldungen an Drittsysteme zu senden und die Daten vom Warenkorb 1:1 in die Datenbank zu schreiben.
Dies hat mit dem heutigen Service Gedanken wenig zu tun und aus architektonischer Sicht wünscht man sich dieses Konzept schnellstmöglich abzulösen. Aus der bereits genannten Anforderung, die Client Schnittstellen nicht zu ändern, ist dies leider nicht möglich und muss nachgebaut werden. Erst in der Phase 2 nach der Inbetriebnahme werden Web Services als Ablösung angeboten. Somit wurde es umso wichtiger, dieses Konstrukt möglichst zu isolieren um andere Teile des Systems nicht zu beeinträchtigen. Dabei wurden folgende Prinzipien angewendet:

- Gleich wie bei den CRUD Schnittstellen werden alle Daten ins kanonische Format transformiert, um auf einfache Weise die neuen Services verwenden zu können.
- Die Warenkorb Management Funktionen wie „anlegen“, Objekte „temporär abspeichern“, „auslesen“ und „löschen“ wurde als eigener Service definiert.
- Die Validierungen sind als weiterer Service realisiert. Die Validierung von einzelnen Entitäten wird dabei auch von den CRUD Operationen verwendet.
- Die Bestimmung des auszuführenden Geschäftsfalles ist mit Oracle Business Rules realisiert.
- Die einzelnen Geschäftsfälle sind als Services implementiert und so konstruiert, dass die Logik auch von den neuen Web-Services wiederverwendet werden kann. (Adapter Pattern).

Architektur Entscheidungen
Aus den bereits geschilderten Herausforderungen mit den Client Schnittstellen wurden bereits ein paar Architektur Entscheidungen ersichtlich. Auch wenn die Schnittstellen einen grossen Einfluss auf die Architektur darstellen, wurde in diesem Projekt sehr darauf geachtet, dass dies auf die Integrationskomponente beschränkt bleibt. Ein weiteres Projektziel ist die Realisierung einer zukunftsorientierten Lösungsarchitektur. Die alten Konzepte stehen im Widerspruch dazu und mussten isoliert werden.

Die wichtigsten Grundbausteine der IVZ System Architektur sind:

- Relationales Datenbankmodell.
- OSB als Integrationskomponente.
- Geschäftslogik und Validierungen realisiert mit der SOA Suite.
- Datenservices als EJB3 Komponenten.
- Oracle Coherence für die temporäre Warenkorb Datenhaltung.
- Datenpflegeapplikation als Standard HTML Web Applikation.

Die vereinfachte Systemarchitektur ist nachfolgend in Abbildung 1 dargestellt.

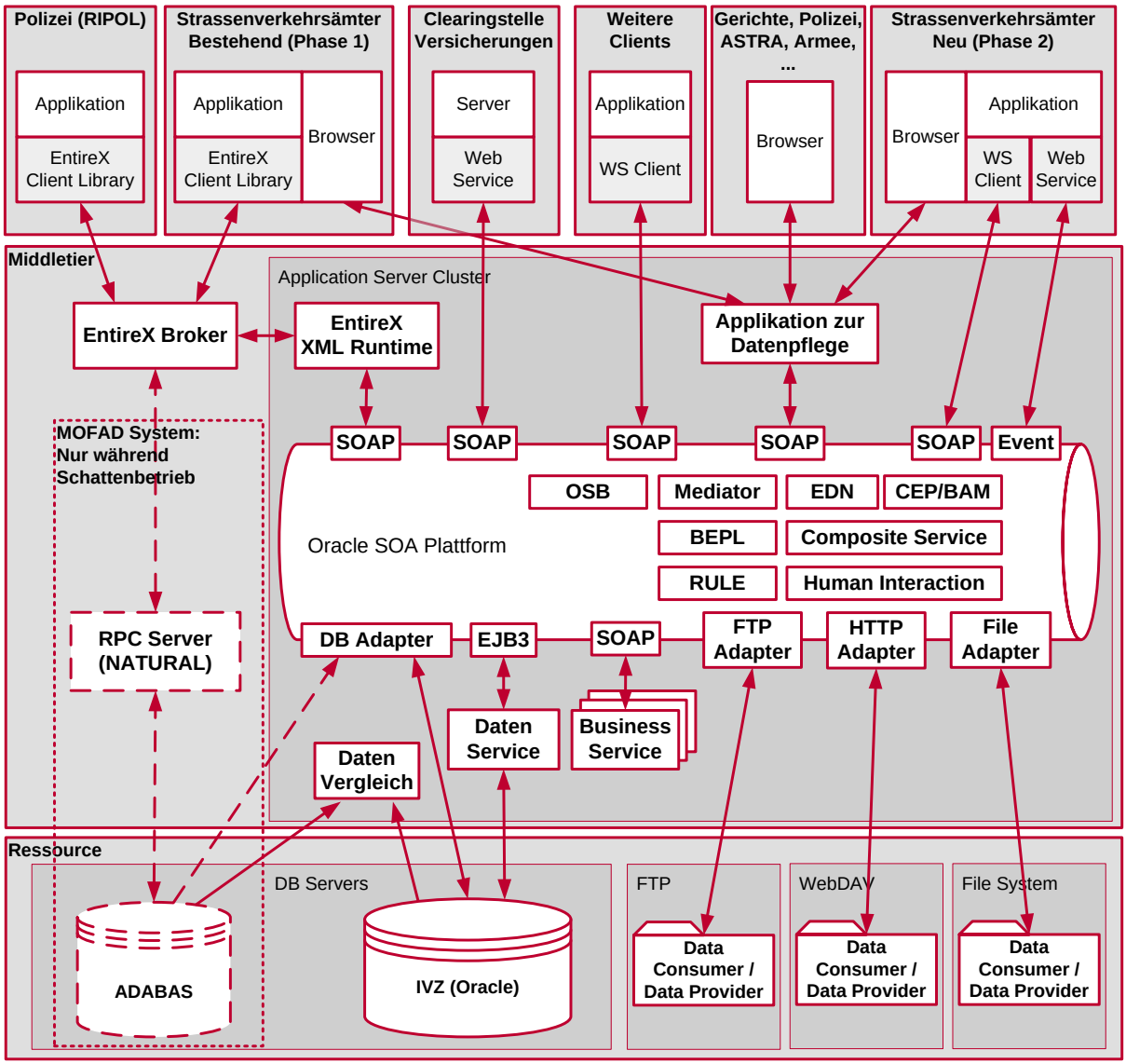

Abbildung 1 IVZ System Architektur

Das Datenbankmodell wurde von Trivadis Modellierungs-Spezialisten in Zusammenarbeit mit Fachexperten neu entwickelt. Einerseits galt es die Beschränkungen aufzuheben und die Möglichkeit zu schaffen Daten zusammenzuführen, wie auch den bestehenden Datenbestand zu übernehmen. Die

phonetische Suche ist mit Oracle Text realisiert, welche gegenüber der Eigenentwicklung im Altsystem eine wesentliche Verbesserung darstellt.

Der OSB übernimmt die Hauptaufgabe als Integrationskomponente. Alle ein- und ausgehenden Schnittstellen laufen über den OSB. Interne Services können ausserhalb IVZ nicht aufgerufen werden, wie auch die internen Services keine externen Services aufrufen können. Die Service Virtualisierung dient der höheren Flexibilität und Testbarkeit, sowie dem zentralen Monitoring und Audit.

Die reinen Datenservices werden als Java EE Komponenten mit Spring, JPA 2 und Hibernate realisiert. Die Anbindung an den OSB erfolgt über EJB3. Dies ermöglicht den Einsatz von Transaktionen bei der Service Orchestrierung und eine sehr hohe Performance, kombiniert mit einer klaren Trennung mit definierter Schnittstelle. Auf Service Data Objects (SDOs) wurde verzichtet, da diese einerseits relativ neu in der Oracle Fusion Middleware sind (seit 11gR1) und anderseits die *Contract First* Philosophie schwieriger zu implementieren ist. Mit der momentanen Implementierung werden die Datenbankstrukturen exponiert, was wir mit unserer Architektur vermeiden wollten. Der Einsatz von EJB3 Datenservices ist auch ein pragmatischer Entscheid, da es sich um wichtige Kernservices handelt. Der Einsatz von bewährten Frameworks und das Vorhandensein von fundiertem Knowhow ist dafür sicher nicht fehl am Platz.

Die Geschäftslogik wird mit der SOA Suite in Business-Process und Business-Activity Services implementiert. Dabei kommen vorwiegend der Mediator, BPEL und Business Rules zum Einsatz.

Als Ablösung für die 3270 Terminals kommt eine webbasierte Datenpflege Applikation zum Einsatz. Die Realisierung erfolgt mit Spring und JSF 2. Die Kommunikation mit den benötigten Web Services erfolgt wiederum über den OSB.

Validierungskonzepte

Im Altsystem sind die Validierungen von Entitäten von zentraler Bedeutung. Dies ist auf folgende Faktoren zurückzuführen:

- Es gibt nur CRUD Schnittstellen und keine richtigen Business Services.
- Viele Datenfelder sind nicht typisiert und erlauben falsche Eingaben durch Charakterfelder für rein numerische oder Datumsfelder.
- Die Middleware beinhaltet nur minimale Business Logik und übernimmt vorwiegend Daten Validierungen.

Eine programmatische Validierung wie im Altsystem nachzubauen stand ausser Frage. Die Ziele von Trivadis sind eine logisch aufgebaute und leicht verständliche Implementierung, gute Wartbarkeit und einfach zu Testen.

Die Schemavalidierung stellte sich augenblicklich als der falsche Ansatz heraus. Auch in Kombination mit weiteren Validierungsarten konnte sie nicht für die alten Schnittstellen eingesetzt werden. Einerseits waren die generierten WSDL Dateien aus den EntireX IDL Definitionen schlicht nicht brauchbar und andererseits mussten feingranulare Validierungs-Fehlermeldungen nachgebaut werden.

Das Problem mit den WSDL Dateien waren immer wiederkehrende anonyme Typendefinitionen. Gleiche Datenstrukturen wurden nicht wiederverwendet, sondern jedes Mal neu definiert anstatt einmal als globalen Typ zu definieren. Zudem wurden alle Datenfelder mit einer statischen Länge eingeschränkt, so dass z.B. ein Textfeld von 30 Zeichen immer genau so lang sein muss oder numerische Felder immer mit Nullen aufgefüllt sein müssen. Die Konvertierung, De-Duplizierung und Globalisierung der anonymen Typen konnte mit XSL Transformationen gelöst werden. Dies war ohnehin notwendig, um die Schnittstellen sauber mit dem OSB einzubinden und Transformationen ins neue kanonische Datenformat durchzuführen. Das Anpassen der einzelnen Felddefinitionen wurde unterlassen, da dies ein zu grosses Risiko darstellte und einen noch wichtigeren Punkt nicht löste: die

feingranularen Validierungsmeldungen. Für zwei Schnittstellen mit gegen 80 Operationen sind über 500 verschiedene Fehlermeldungen definiert, welche übernommen werden müssen!
Hier kommt man mit XSD nicht sehr weit, man kann zwar herausfinden, dass die Datenstruktur fehlerhaft ist, aber vielfach nicht welches Feld warum nicht gültig ist.

Die Validierungsproblematik liesse sich vollständig mit Oracle Business Rules lösen. In der ersten Prototyp Phase zeigte sich aber schnell, dass dies schlicht Overkill ist und der Implementierungsaufwand nicht gerechtfertigt ist. Der Grossteil aller Validierungen ist relativ simpel, wie Feldlängenprüfungen und Datentypvalidierungen. Für dies muss keine Rule Engine eingesetzt werden. Es ist sinnvoller nur die komplexen und dynamischen Validierungen der Rule Engine zu überlassen. Ein weiterer Vorteil ist die vereinfachte Datenhandhabung in der Oracle Business Rules, wenn die Daten bereits auf Datentyp- und Feld-Ebene validiert wurden. Dies erspart viele zusätzliche Prüfungen und bläht die wirklich wichtigen Regeln nicht zusätzlich auf.

Schematron
Als Lösung stellte sich Schematron heraus. Im Gegensatz zu XSD kann damit der Inhalt von XML Dokumenten validiert werden. Es können komplexe Regeln definiert werden, welche Abhängigkeiten von verschiedenen Teilen des Dokumentbaums umfassen. Ein klassisches Beispiel sind Datums-Abhängigkeiten, so dass z.B. ein Geburtsdatum nicht in der Zukunft liegen darf oder ein Bestelldatum vor dem Lieferdatum liegen muss.
Eine weitere Stärke liegt im Reporting der Validierungsfehler. Für jede Regel wird ein Fehlertext definiert, welcher bei Verletzung der Regel zurückgeliefert wird. Der Fehlertext ist in unserem Fall ein zusammengesetzter Fehlercode und nicht bereits die an den Client zurückgelieferte Fehlermeldung. Der Fehlercode besteht aus Fehlerart, Entitätsname, Feldname und einem Fehlertyp und dient hauptsächlich der besseren Lesbarkeit. Rein technisch wird dieser als ein Code angeschaut und wird anhand einer DB Mapping Tabelle in die vier Landessprachen sowie den Legacy System spezifischen Error Codes übersetzt.
Nachfolgend ein stark vereinfachtes Beispiel aus der Personenvalidierung mit Schematron.

```
  <pattern name="Field tests">
    <rule context="/n1:ValidatePersonRequest/dat:Person">
[1]   <assert test="xp20:matches(dat:LastName, '.{1,30}')">
               [E|Person|LastName|INVALID]</assert>
[2]   <assert test="xp20:matches(dat:Language, '(D|F|I|R)')">
               [E|Person|Language|INVALID]</assert>
[3]   <assert test="not(dat:BirthDate/node()) or dat:BirthDate &lt;
        xp20:current-date()">[E|Person|BirthDate|NOT_IN_PAST]</assert>
[4]   <assert test="not(dat:Gender='J') or dat:Gender ='J' and not
       (dat:BirthDate/node())">[E|Person|BirthDate|NOT_ALLOWED]</assert>
    </rule>
  </pattern>
```

Erläuterungen:
- [1] Der Nachnamen muss 1-30 Zeichen umfassen
- [2] Die Sprache muss D, F, I oder R sein
- [3] Wenn ein Geburtsdatum vorhanden ist, muss dieses in der Vergangenheit liegen
- [4] Bei juristischen Personen darf kein Geburtsdatum gesetzt sein

Leider wird das 'report' Element nicht unterstützt, um negative Regeln zu definieren. D.h. wenn die Regel gültig ist, wird die Meldung ausgegeben. Regel [3] und [4] könnten damit vereinfacht werden:

```
  <assert report="dat:BirthDate > xp20:current-date()">…</assert>
  <assert report="dat:Gender ='J' and dat:BirthDate/node()">…</assert>
```

Die Daten Validierung wurde als Validierungs-Service mit einem SOA Composite realisiert. Damit konnte die Schematron Validierung im SOA Mediator eingebunden werden, welcher diese direkt unterstützt. Einzig die Transformation der Fault Meldung bei Validierungsfehlern musste noch im OSB vorgenommen werden. In einem weiteren Schritt werden, wie bereits angedeutet, die IVZ Fehlercodes in die alten Fehlerstrukturen und der benötigten Sprache gemappt.

Oracle Business Rules

Die Oracle Business Rules (OBR) wurde mit dem 11g Release eine zentrale SCA-Komponente. Sie kann mit XML- und Java-Fakten umgehen und in SOA/BPM Komponenten wie auch unabhängig als Web-Service und direkt in Java eingesetzt werden.

Die OBR werden für alle Validierungen eingesetzt, welche mit Schematron nicht mehr abgebildet werden konnten oder deren Definition schlicht zu aufwendig wurde.

Im IVZ System werden die komplexeren Plausibilisierungen mit mehreren abhängigen Datenfeldern, sowie die fachlichen Querprüfungen zwischen Entitäten in der Rules Engine abgebildet. Eine der wichtigsten Regeln, die vom Legacy System übernommen und nachgebildet werden mussten, sind altersabhängige Validierungsregeln. Anhand wann und von welchem System eine Entität erstellt wurde, wird sie in bis zu fünf Alterskategorien klassifiziert. Die Kategorie bestimmt danach, welche Validierungen durchgeführt werden müssen.

Eine Rules Engine ist für eine solche Aufgabe prädestiniert. In einem Regelset wird die Alterskategorie ermittelt und daraus ein neues Faktum erstellt. Dieses löst danach die zu testenden Regeln aus. Eine Validierungsregel wird nur durchlaufen, wenn sie die entsprechende Alterskategorie in der Eingangsbedingung erfüllt.

Die Abbildung 2 zeigt die Definition der Altersklassenbestimmung anhand einer „Decision Table“. Ein Vorteil von OBR sticht mit diesem Beispiel bereits heraus: Die visuelle tabellarische Darstellung führt zu einer kompakten und schnell verständlichen Übersicht. Mit einer klassischen programmatischen Implementation ist viel mehr Aufwand erforderlich, die Abläufe zu verstehen und anzupassen.

Conditions		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
C1	out.conversionResult.vehicle....	otherwise									"X"
C2	out.conversionResult.vehicle....	Ausserverkehr			otherwise,Inverkehr						-
C3	out.conversionResult.vehicle....	-			...t,Ausland	-					-
C4	VehicleInFact.recordKind	N	A	X,U,C	-	N	X,U		A	C	-
C5	after(VehicleInFact.vehicle.re...	-	-	-	-	-	true	false	-	-	-
Conflict Resolution											
	Override				R5, R6, R...						
Actions											
A1	modify out.validationResult...	☑	☑	☐	☑	☑	☑	☑	☑	☑	☑
	validationClass:DataValid...	....CLASS_1	....CLASS_4	....CLASS_3	...CLASS_3	....CLASS_1	...CLASS_2	....CLASS_3	...CLASS_4	....CLASS_5	...CLASS_3
A2	call RL.assert a tree of fact...	☑	☑	☐	☑	☑	☑	☑	☑	☑	☑
	root:Object)	out	out	out	out	out	out	out	out	out	out

Abbildung 2 Oracle Business Rules zur Bestimmung der Alterskategorie

Schattenbetrieb

Die Kundenanforderungen umfassen einen asynchronen Schattenbetrieb, um das korrekte Verhalten des Systems nachzuweisen. Fehlfunktionen beeinträchtigen alle Strassenverkehrsämter der Schweiz, die Polizei und weitere abhängige Systeme – auch in der EU. Daher steht die Risikominimierung beim Kunden verständlicherweise an oberster Stelle.

Gegenüber einem synchronen Parallelbetrieb bietet dies folgende Vorteile:

- Das Altsystem bleibt als Master erhalten und das Systemverhalten ändert sich nicht. Erst bei Erreichen der definierten Fehlerquote und stabilen Betrieb wird umgeschaltet.

- Durch die asynchrone Entkoppelung kann es zu keinen aussergewöhnlichen Verzögerungen kommen. Die Antwortzeiten verlängern sich nur um das Routing durch den Meldefluss-Recorder.
- Der Meldefluss kann zum mehrfachen Ausführen in der Datenbank gespeichert werden, oder nur über Queues laufen, wenn die Meldungen direkt im IVZ System ausgeführt werden sollen.
- Der Ablauf ist reproduzierbar. D.h. wenn Fehler entdeckt und korrigiert wurden, kann der ganze Meldefluss wieder auf der Abnahme Umgebung abgespielt werden ohne einen weiteren produktiven Test zu initiieren.
- Zusätzlich lassen sich damit bereits während der Implementierungsphase Lasttests auf den einzelnen Infrastruktur- wie auch System-Komponenten durchführen.

Die Funktionsweise des Schattenbetriebes ist schematisch in Abbildung 3 dargestellt.

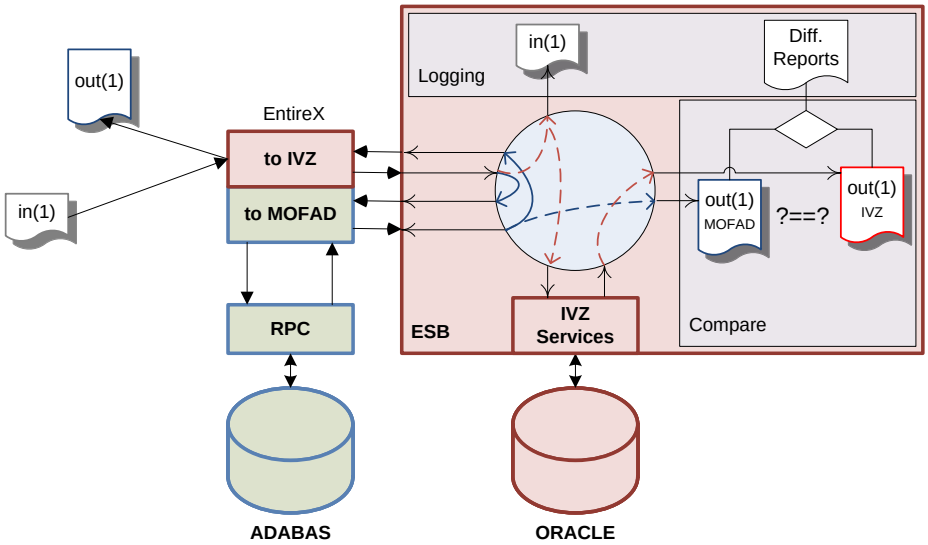

Abbildung 3 Funktionsweise des Schattenbetriebes

- Die Client Meldungen des Altsystems werden über den OSB geroutet. Dies gilt für die früher erwähnten EntireX Meldungen, welche in SOAP Aufrufe konvertiert wurden, wie auch für weitere Web Service Schnittstellen des Altsystems.
- Der Meldeverkehr (Request und Response Meldungen) werden in eine Queue geschrieben und von IVZ asynchron in der Datenbank persistiert. Dies erlaubt eine sofortige Weiterleitung der Response Meldung an den Client, ohne auf die Persistierung zu warten.
- Das zeitversetzte Ausführen der Meldungen auf IVZ kann gesteuert werden, entweder direkt nachdem die Response Meldung von MOFAD erfolgt, oder eine beliebige Zeit später durch erneutes Auslesen der Meldungen aus der Datenbank
- Ein Vergleichsservice wertet beide Antwortmeldungen aus und schreibt das Resultat in die DB. Ein weiteres Analyse Tool generiert zusätzliche Reporte und führt Datenvergleiche auf Datenbankebene durch.

Fazit
Das DM2010 Projekt befindet sich zwar noch vor der Realisierungsphase, aber es kann dennoch bereits ein umfassendes Fazit gezogen werden. Vor allem durch die zwei Prototyp Phasen, in welchen sich die definierte Architektur bewähren und Performance Nachweise erbracht werden mussten.

Was sich bewährt hat:

- Contract First Design. Die Gefahr besteht überall die WSDLs zu generieren und die XML Schemas mit unterschiedlichen Konzepten zu definieren. Hier ist Disziplin gefragt und das Vorgehen nach Best Practice Richtlinien.
- Kontinuierliche Integration von Anfang an einzusetzen. Dies kann anfänglich einen sehr grossen Aufwand bedeuten, wenn noch kein Knowhow existiert. Es zahlt sich aber mehrfach aus. Siehe DOAG Vortrag „CI for SOA“!
- soapUI is your friend! Ein unentbehrliches Tool wenn es ums Testen und Automatisieren geht.
- Mächtige Rule Engine direkt als SCA Komponente in der SOA Suite verwendbar, ohne zusätzlich etwas installieren bzw. lizenzieren zu müssen. Der Umgang mit der Rule Engine zur Entwicklungszeit ist wesentlich komfortabler als noch mit SOA Suite 10g.
- JMS Queues direkt in Oracle WebLogic verfügbar und sehr einfach in OSB integrierbar bzw. vom OSB verwendbar. Es braucht keine zusätzliche MOM Komponente.
- Oracle Web Service Manager als SOA Security Lösung. Security Policies lassen sich deklarativ angeben und zentral verwalten.
- XSLT Mapping Editor in der SOA Suite. Beim manuellen Bearbeiten der XSLT Dateien müssen wesentlich weniger Einschränkungen in Kauf genommen werden, als mit dem OSB XQuery Editor, damit der visuelle Editor weiterhin verwendet werden kann.

Herausragende OSB Eigenschaften:

- Sehr mächtige und performante Integrationskomponente.
- XML Transformationen (XQuery & XSLT) und Service Virtualisierung.
- Effiziente Entwicklung (wenn die Architektur und Interfaces im Voraus definiert werden!) und komfortable Debugging Möglichkeiten.
- Das Laufzeit-Monitoring und –Tracing erlaubt Probleme schnell zu lokalisieren.

Was sich nicht bewährt hat:

- Möglichst alle Regeln in einer einzigen Oracle Business Rules Komponente zu definieren. Darunter leidet die Übersichtlichkeit und Handhabung. Ausserdem sind Änderungen an den Input- und Output-Facts Definitionen heikel und muss vorsichtig durchgeführt werden um die definierten Regeln nicht zu beeinträchtigen. D.h. es empfiehlt sich mehrere OBR Komponenten einzusetzen und unabhängige Regeln zu separieren.
- OBR direkt via Web Service Schnittstelle zu exponieren. Ist zwar praktisch, widerspricht aber der *Contract First* Philosophie. Deshalb besser mit Mediator und Transformationen verwenden.
- Zu wenig leistungsstarke Laptops als Entwicklungsrechner einzusetzen. Eine moderne CPU mit möglichst hoher Taktfrequenz kombiniert mit einer SSD spart enorm viel Zeit und Nerven. Vor allem die OSB Entwicklungsumgebung ist nur bedingt Multithreading fähig und profitiert am meisten davon. Auch die Startzeiten von WebLogic Servern darf nicht unterschätzt werden.

Neben all den positiven OSB Eigenschaften gibt es auch Schwachpunkte:

- Ein übermässiger Einsatz der OSB Split-Join Komponente: Error-Handling unterscheidet sich gegenüber Proxy Services. Zudem verhält sich die Entwicklungsumgebung damit sehr träge.
- Die Entwicklungsumgebung bietet keine Test-Unterstützung und bietet nicht den Funktionsumfang des Web-Interfaces (z.B. Service Monitoring und SLA Alerts).
- Das EJB zu XML Mapping ist nicht konfigurierbar (JAXB wäre z.B. wünschenswert).
- Keine Unterstützung für XQuery Module und XSLT 2.0. Dies wird durch BEA Erweiterungen zwar grösstenteils wettgemacht, jedoch werden in der SOA Suite Oracle spezifische Erweiterungen verwendet.

Kontaktadresse:

Markus Zehnder
Trivadis AG

Papiermühlestrasse 73	Tel:	+41(0)31-928 09 60
CH-3014 Bern	Fax:	+41(0)31-928 09 64
Internet: www.trivadis.com	Mail:	markus.zehnder@trivadis.com