Hinter den Kulissen

Modellrepository @ T-Mobile

Carsten Sensler, Andre Karalus, Markward Märtens

Seit 2006 wird bei der T-Mobile eine große SOA-Infrastruktur (SOA-Backplane) europaweit betrieben (mehr als 3 Millionen Service-Calls/Tag). Herzstück zur Designtime ist ein Modellrepository namens CEISeR, in welchem die gesamte SOA-Architektur der T-Mobile aufgenommen und verwaltet werden soll. Zurzeit sind ca. 1400 Service-Definitionen sowie ca. 2500 Kommunikationsbeziehungen in CEISeR hinterlegt. In diesem Artikel wird dargestellt, mit welchen Konzepten die T-Mobile ein Modellrepository umgesetzt hat, und es wird erläutert, wie sich die konkrete Umsetzung für die Domäne „serviceorientierte Architektur" ausgestaltet; das daraus resultierende domänenspezifische Modellrepository ist CEISeR.*

Was ist ein Modellrepository?

Ein Modellrepository dient der Beschreibung von Daten- bzw. Objektmodellen. Im allgemeinen Fall geht es genauer darum, ein Objektmodell, welches ein System beschreibt, zu persistieren.

Kann das nicht auch ein beliebiger O/R-Mapper wie z. B. JPA/Hibernate und ist ein Repository nicht nur eine Datenbank? Nein, bei einer objektorientierten Datenbank sind einige der unten beschriebenen nichtfunktionalen Anforderungen, wie z. B. die Darstellung der zeitlichen Entwicklung des Systems und gleichzeitig gültige unterschiedliche Versionen, nicht erfüllt und sind definitiv in einer Schicht darüber zu realisieren. Die Zusammenfassung dieser Schicht mit der darunter liegenden Persistierungstechnologie ist in unserem Fall in der Core-Plattform zusammengefasst.

Das Modellrepository @ T-Mobile

Zu Beginn des Projektes wurde analysiert, welche Möglichkeiten der Modellierung eines Repositorys bestehen und wie der Quellcode dafür erstellt werden kann. Eine wichtige Anforderung, die nach anfänglichen Diskussionen um das Modell klar wurde, war Flexibilität! Da die Fachlichkeit für das Modellrepository noch nicht komplett spezifiziert war – schließlich gab es bisher nichts Vergleichbares –, sollte das Repository flexibel bezüglich Änderungen des fachlichen Modells sein.

Der gewählte Ansatz der modellgetriebenen Softwareentwicklung hat sich in der Retrospektive optimal bewährt. Als Werkzeug wurde dabei zur UML-Modellierung „Magic Draw" gewählt, als weitere Technologie kamen „Eclipse EMF" (EMF=Eclipse Modeling Framework) und das MDSD-Generatorframework „openArchitectureWare" [oAW] zum Einsatz. Mit Magic Draw wird dabei das fachliche Domänenmodell editiert und anschließend einem oAW-Generator-Workflow übergeben, welcher durch ein Ant-Skript angestoßen wird.

Abbildung 1 gibt einen Überblick über den oAW-Generator-Workflow für das domänenspezifische Modellrepository CEISeR (**C**entral **E**nterprise **I**ntegration **Se**rvice **R**epository):

1. *mdumltrafo*: Erzeugt das eiCommon.mmgen-Metamodell aus dem Magic-Draw-UML2-Modell (Modelltransformation). Das eiCommon.mmgen ist selbst ein Modell des mmgen.ecore-EMF-Metamodells, welches als Ausgangsbasis für weitere Generatoren dient.
2. *mmgen.generator*: Erzeugt aus dem eiCommon.mmgen den CEISeR-Core.
3. *gmf.generator*: Erzeugt das CEISeR.xmetamodel, ein weiteres Metamodell, das für das Importer-Framework und den gra-

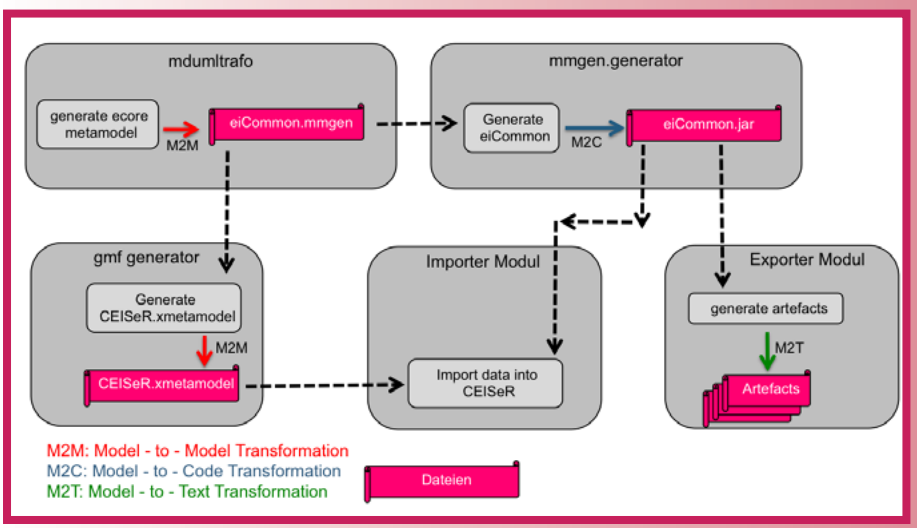

Abb. 1: Zusammenhang der Generatoren

* Designtime: Modellierung der SOA-Architektur, d.h. im Wesentlichen das Modellieren der Kommunikationsbeziehungen zwischen Service-Anbieter und Service-Nutzer getrennt nach Anwendungen und Komponenten; weitere Informationen unter [SeKa08a,SeKa08b,SeKa08c].

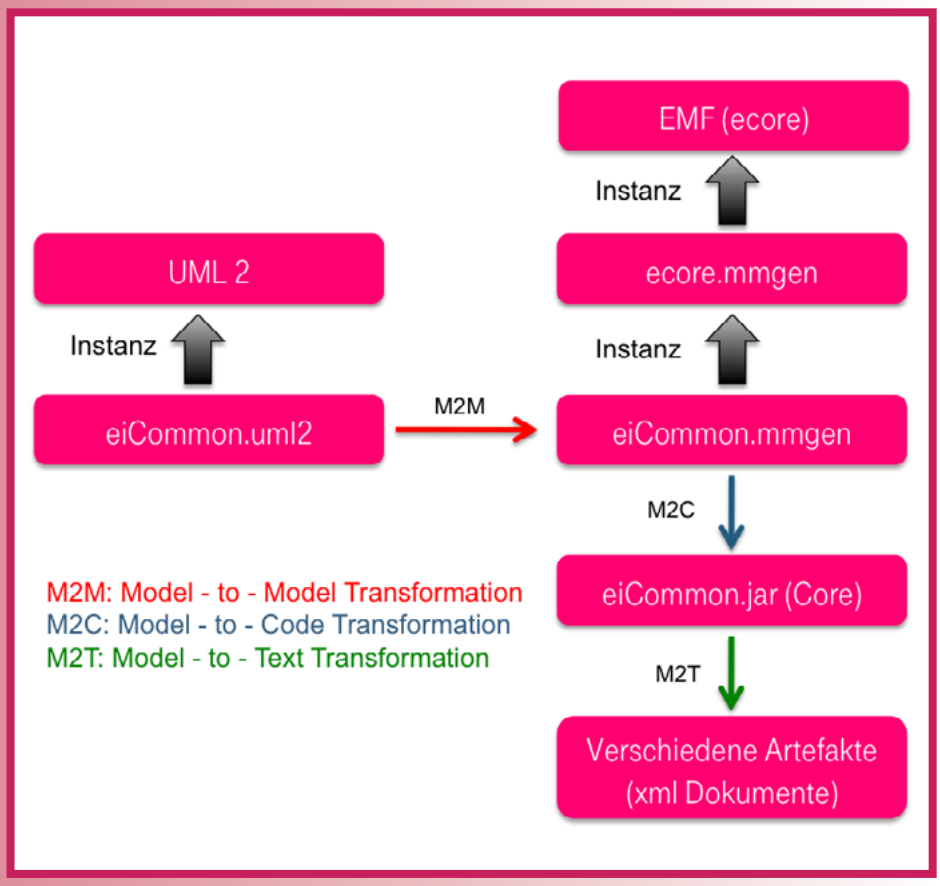

Abb. 2: (Meta)Modelle und deren Zusammenspiel

fischen Editor Xplor genutzt wird. Die oAW-Templates setzen auf dem eiCommon.mmgen auf.

4. *Exporter Modul*: Erzeugt aus dem CEISeR-Modell mittels Instantiierung des Objektmodells unter Nutzung der core.platform (Cores) Artefakte für die Laufzeitumgebung (XML-Konfigurationsdateien). Die oAW-Templates nutzen eiCommon.jar mit dem JavaBeans-Adapter.
5. *Importer Modul*: Nutzt das CEISeR.metamodel, um Adapter-Klassen zu generieren, und benötigt ebenfalls das eiCommon.jar.

Abbildung 2 verdeutlicht den Zusammenhang der verschiedenen Modelle und deren Transformationen.

Für die Modellierung des Domänenmodells wurde in Magic Draw ein UML2-Profil angelegt, das für die weitere Verarbeitung bestimmte Stereotype und tagged Values definiert. Das Modell wird im EMF-UML2-Format exportiert, damit es in oAW leicht instantiiert werden kann. Die Model-to-Model-Transformation (M2M) und das Zwischenmodell eiCommon.mmgen erleichtern die Erstellung der weiteren M2M-Transformatoren und Generatoren, da sich dieses auf die dafür wesentlichen Informationen beschränkt. Auch wird bei dieser M2M-Transformation ein Postprocessing durchgeführt, bei welchem dem Modell weitere Informationen zugewoben bzw. Default-Attribute gesetzt werden, die sich aus Eigenschaften und Beschränkungen der Plattform ergeben. Das Metamodell mmgen.ecore ist selbst allerdings noch völlig unabhängig von dem domänenspezifischen Modell CEISeR, es beschreibt lediglich das generelle Konzept von Entitäten und Referenzen dazwischen, ohne die volle Mächtigkeit von UML2 zu benutzen.

Der Core-Generator

Zunächst einmal gehen wir auf den zweiten Schritt im oAW-Generator-Workflow ein. Die oAW-Templates im mmgen.generator erzeugen für alle modellierten Domänenklassen (mit Stereotyp Entity) Folgendes:

- ein Entity-Interface, deklariert alle Getter und Setter für Attribute und Referenzen,
- eine abstrakte Entity-Klasse, implementiert Getter und Setter - enthält plattformspezifischen Code,
- eine konkrete Entity-Klasse für manuell zu erstellenden Code,
- ein Hibernate-Mapping zur Persistierung der Daten in einer Datenbank mittels Hibernate.

Die konkrete Entity-Klasse enthält zunächst keinen Code bzw. nur Methodenrümpfe, sie wird in ein Verzeichnis „src-once“ generiert, und damit bei nachfolgenden Generatorläufen nicht mehr überschrieben (so geht der Implementierungscode nicht mehr verloren). Dies ist ein übliches Vorgehen bei der Arbeit mit oAW und ersetzt die Definition von Protected Regions in XPAND, der Template-Sprache von oAW.

Alle anderen Klassen werden in „src-gen“ geschrieben – sie werden bei jedem Generatorlauf mit den neuen Versionen überschrieben, entweder weil sich das fachliche Modell geändert hat, oder weil die oAW-Templates zur Unterstützung der Plattform geändert worden sind.

Weitere Klassen, die für das gesamte Modell generiert werden, sind

- HibernateConfiguration: instantiiert alle Hibernate-Mappings,
- HibernateSchemaHelper: legt Datenbank-Schemata an,
- MetaModelHelper: stellt Meta-Informationen zum Modell zur Verfügung,
- DBCloner: Migrationstools zur Migration vorheriger Modellrepräsentationen aus einer Datenbank in eine andere.

Die Core-Plattform

Die Plattform enthält den nicht entitätenspezifischen Code, der zur Nutzung des generierten Domänenmodells (eiCommon.jar) nötig ist. Die generierten Entity-Klassen setzen auf der Plattform auf.

Wichtige Konzepte

Die Entity-Klassen bilden eine Hierarchie aus Parent-/Child-Klassen; die Containment-/Composition-Referenzen aus dem Domänenmodell definieren die Child-Beziehung. Alle Toplevel-Entity-Klassen (Root in einer Parent-/Child-Hierarchie) besitzen eine Beziehung zu einer ausgezeichneten (und bei der M2M-Transformation dazugewobenen) Namespace-Klasse. In dieser Hierarchie lassen sich dann alle Elemente

mit einem voll qualifizierten Namen (FQN) nach dem Muster „namespace1.namespace2.root:child:grandchild“ adressieren. Dieser FQN ist ein fachlicher Schlüssel, für den es in der Plattform Lookup-Methoden gibt.

Die physische Chronologie bedeutet, dass Entitäten und ihre Referenzen versioniert werden. Jede Entität wird in einer bestimmten Transaktion angelegt und behält ihr Gültigkeit bis zu der Transaktion, in der sie gelöscht wird. Es lässt sich ein früherer Versionsstand des Repositorys referenzieren (im einfachsten Fall mit einer Transaktionsnummer) und der damalige Zustand wieder auslesen. Außerdem können in einem Report die Änderungen an einer Entität über die Zeit dargestellt werden. In der Persistenzschicht werden daher nie Daten gelöscht, sondern nur neue Versionen angelegt. Die Gültigkeitszeiträume werden dabei an den Transaktionsnummern festgemacht.

Alle Zugriffe auf die Entitäten erfolgen über einen Modellkontext (ConnectionToken-Klasse in core.platform), bei der Erzeugung kann man den Kontext (Transaktionsnummer) angeben, auf den sich das Lesen der Entitäten bezieht. Beim Schreiben werden Änderungen unter einer neuen, global eindeutigen Transaktionsnummer gespeichert. Die CRUD-Aktionen, die in einer Transaktion mit einem ConnectionToken gemacht werden, sind mittels physischer Chronologie vollständig von denen anderer Transaktionen abgeschirmt [Sun].

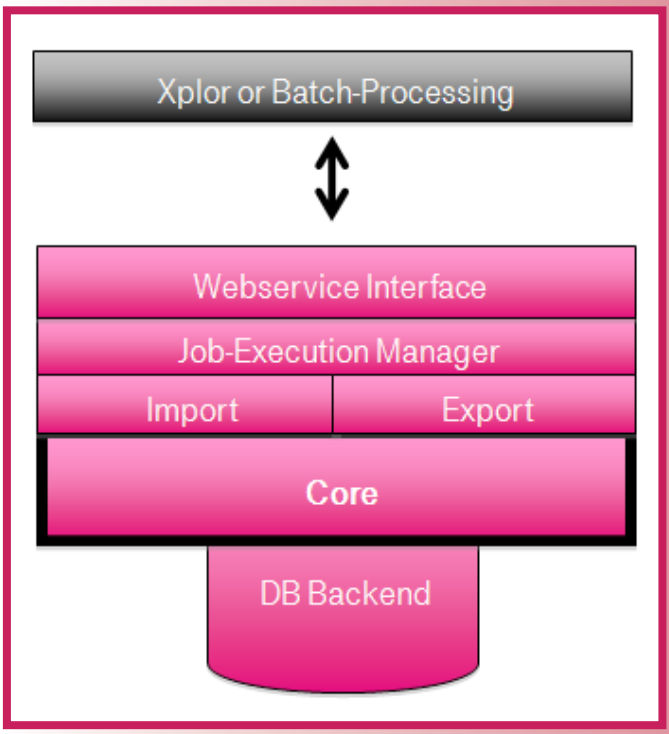

Abb. 3: Architektur-Überblick - CEISeR

Die funktionalen Anforderungen

Die Plattform erfüllt folgende wesentlichen Anforderungen:

- Generierung von Zugriffsklassen aus dem Domänenmodell für die Benutzung in den darauf aufsetzenden Layern. Die Zugriffsklassen ermöglichen die transparente Persistenz und stellen Getter/Setter für Attribute und Beziehungen bereit. Die Getter/Setter für Beziehungen sorgen für Konsistenz bei bidirektionalen Referenzen. Für Kompositionsbeziehungen wird der Lifecycle verknüpft, d. h. wird ein Elternteil gelöscht, dann auch die entsprechenden Kinder.
- Proxykonzept, d. h. für Entitäten, deren konkrete Ausprägung zum Zeitpunkt einer Transaktion noch nicht bekannt ist, die aber aufgrund von Kardinalitäts-Constraints existieren müssen, können Proxys angelegt werden. Diese Proxys können dann in einer späteren Transaktion transparent durch richtige Entitäten ersetzt werden, bzw. werden automatisch gelöscht, wenn keine Referenz mehr darauf verweist.
- CRUD-Operationen und ein Query-API für freidefinierbare Suchen über Reference-Namen und Attribute auch unter Berücksichtigung von Vererbung zwischen Entitys.
- Verschiedene Cache-Strategien: Eager und Lazy Loading sind stufenweise konfigurierbar.
- Transaktionen mit Rollback/Commit: Bei einem Commit werden modellierte sowie implementierte Constraints automatisch überprüft. Diese Constraints sind für verschieden Use-Cases konfigurierbar. Modellierte Constraints werden aus dem Domänenmodell generiert, z. B. für Zu-1-Kardinalitäten. Bei der Konfiguration per Use-Case wird z. B. festgelegt, dass beim Import ein vorhandener Proxy eines Entity-Typs nur eine Warnung ergibt, beim Generieren der Laufzeitkonfiguration derselbe Constraint aber einen Fehler produziert.
- Optimistisches Sperren für parallele Transaktionen.
- Erstellen von Labels für bestimmte Versionen des Repositorys (statische Sicht), sowie Erzeugen von Branches bestimmter Stände, auf denen dann unabhängig vom Hauptstrang Änderungen vollführt werden können. Unterstützung von „merge“-Transaktionen.
- Autorisierung für schreibende Änderungen an Entitäten basierend auf dem Namespace. Dazu implementiert die Plattform ein Benutzer/Rollen-Konzept. Es ist eine Unterstützung für die Autorisierung anderer Aktionen ebenfalls vorhanden. Für die Autorisierung zur Veränderung von Beziehungen werden im Domänenmodell Assoziationen als „primär“ dekoriert.
- Dazuweben nicht funktionaler Entitäten; genutzt für Statusmodelle, d. h. Entitäten können einen Status aus freidefinierbaren Statusmodellen besitzen; Entitäten können einen Contact besitzen (z. B. für E-Mail-Notifizierung bei Änderungen an einer Entität).
- Beliebige key/value-Property an Entitäten, sowie Beschreibungsfelder für Dokumentation usw.
- Ein generisches Zugriffs-API, mit dem à la Reflection-API auf die Entitäten zugegriffen werden kann.

Unterstützung Konfigurationsmanagement

Durch das Spezifizieren eines Kontextes bei lesendem Zugriff auf die Daten lassen sich beliebige Zustände des Repositorys referenzieren. Dieser Kontext kann aus nur einer Transaktionsnummer bestehen oder auch aus verschiedenen Transaktionsnummern für verschiedene Bereiche des Modellrepositorys. Mit dem Kontext kann sich der Anwender eine beliebige statische Sicht auf das Repository definieren. Zusätzlich kann der Kontext mit einem Label (symbolischer Name) versehen werden, z. B. nach einer Produktivstellung von neuen Kommunikationsbeziehungen wird das Label *ProduktionQ4_08* (basierend auf Transaktionsnummer 4711) erstellt. Mit dem Label *ProduktionQ4_08* lässt sich in der domänenspezifischen Ausprägung CEISeR zu jeder Zeit der Inhalt von CEISeR für den Zeitpunkt der Produktivstellung erneut darstellen; auch wenn bereits neue Anteile nach CEISeR impor-

tiert wurden oder wenn bestehende Aspekte geändert worden sind. Man erhält eine statische Sicht auf einen bestimmten Stand des Repositorys. Dieser Mechanismus ist vergleichbar mit dem Erstellen eines Tags in Subversion – mit einer Ausnahme, in Subversion sind auch Tags prinzipiell änderbar, in dem Modellrepository nicht.

Von diesem Stand kann auch ein Branch erstellt werden. Ein Branch ist eine zweite Art der statischen Sicht eines definierten Standes im Modellrepository, mit dem Unterschied zu den Labels, dass Branches auch verändert werden können. Zusammenfassend kann man sagen, im Modellrepository unterscheiden wir zwischen nur-lesenden (Labels) und änderbaren (Branches) statischen Sichten; in Subversion gibt es diese explizite Unterscheidung nicht.

Diese Metaphern ähneln sehr stark denen von Versionsverwaltungssystemen, beispielsweise ist die Subversion-Revisionsnummer analog zur Transaktionsnummer aus dem Core. Damit ist eine optimale Unterstützung des Konfigurationsmanagements möglich, wo es nicht ausreicht, nur Code zu versionieren, sondern erforderlich ist, den gesamten Stand der Software, wie sie ausgeliefert wird, zu verfolgen [SeKa08a].

CEISeR - die konkrete Ausprägung

Der Core wird zusammen mit dem CEISeR-Domänenmodell als Anwendung von verschiedenen Komponenten genutzt. Die wichtigsten Anwendungsfälle sind das CEISeR-Import- sowie -Export-Modul. Da es sich von Beginn des SOA-Backplane-Programms an abzeichnete, dass die Anwender von CEISeR Daten in großen Mengen zuliefern (z. B. Dutzende WSDL-Dateien und XSDs) werden, ist ein Job-Framework entwickelt worden, das Jobs abarbeitet und mittels „Tasks“, die in einer Jobbeschreibungsdatei definiert werden, gegen das Repository synchronisiert. Die eigentliche Zugriffsschicht nach außen ist durch eine Webservice-Schnittstelle umgesetzt worden. Jobs können dem Job-Execution Manager entweder durch ein Batchprogramm oder durch den Xplor übergeben werden.

Der Xplor ist eine grafische Applikation basierend auf Eclipse RCP, die es ermöglicht Jobbeschreibungsdateien zu editieren, eine statisches Abbild eines spezifischen Standes von CEISeR zu analysieren, sowie verschiedene Modelle zu erstellen. In CEISeR werden aus Benutzersicht im Wesentlichen folgende Modelle unterschieden:

- Architekturen: Definition von Kommunikationsbeziehungen,
- Infrastrukturen: Physische Laufzeitumgebungen der SOA-Backplane (DEV, TST1, TST2, PRD),
- Bindings: Legt fest, welche Komponenten aus der abstrakten Architektur in welche konkrete Umgebung provisioniert werden; d. h. Freischalten konkreter Kommunikationsbeziehungen in einer spezifischen Laufzeitumgebung.

Diese Modelle können mit dem Xplor grafisch unterstützt editiert werden. Außerdem werden mit dem Xplor die Jobs ausgeführt; auch die Ergebnisse der ausgeführten Jobs (Logs) können wiederum grafisch unterstützt im Xplor analysiert werden. Der Xplor ist aber nicht das einzige Werkzeug, um CEISeR mit neuen Inhalten zu bestücken. Es ist auch eine Batch-Schnittstelle für Windows und Linux vorhanden, damit ein bestehender Entwicklungsprozess oder bestehende Build-Automatismen sich mit CEISeR integrieren können.

Der Importer ist ein Modul, der die Eingangsdaten gegen das Repository synchronisiert und nur Unterschiede zurückschreibt.

Das Exporter-Modul besteht aus einem oAW-Workflow, dem Informationen über zu generierende Services (per FQN) übergeben werden und der dann den Core benutzt, um mit der JavaBeans-Strategie auf die Entitäten zuzugreifen und damit dann die Konfiguration für die Laufzeitumgebung zu generieren. Die Konfiguration ist daraus abgeleitet, also welche Komponenten miteinander in einer Umgebung über welche Schnittstellen (WSDL) kommunizieren. Daraus werden dann WARs/EARs gebaut und über einen weiteren Schritt in die Laufzeitumgebung ausgeliefert [SeKa08b,SeKa08c].

Als weiteren Anwendungsfall gibt es einen dynamischen Report, der als Webanwendung den CEISeR-Core direkt instantiiert und damit ermöglicht, auf dem Repository zu browsen und dabei auch in die historisierten Daten Einblick gewährt. Ebenfalls lassen sich mit dem Report bequem die verschiedenen Branches anwählen und alle Änderung einer Transaktion ansehen (Auditing).

Wiederverwendung des Cores für andere Domänen

Der Core des Modellrepositorys ist grundsätzlich auch für andere Domänen geeignet. Die bisher einzige konkrete Ausprägung des Cores des Modellrepositorys ist CEISeR.

Der Core ist testgetrieben entwickelt worden; so existiert auch ein synthetisches Testmodell mit Klassen wie z. B. Person, Adresse, um eine umfangreiche JUnit-Testsuite ablaufen lassen zu können, welche die Kernfunktionalitäten des Cores ohne das domänenspezifische CEISeR-Modell regressionstestet. Auch in anderen Bereichen der T-Mobile (anderen Domänen) ist seine Verwendung denkbar.

Fazit

Ist der modellgetriebene Ansatz, der in diesem Projekt konsequent verfolgt wird, der richtige? Bezogen auf die Anforderungen und auf die zeitlichen Rahmenbedingungen, ist es der richtige Weg. Initial war es sicherlich etwas mehr Aufwand, die verschiedenen Transformationen durch Generatoren umzusetzen, aber mittlerweile ist die Maschinerie soweit fortgeschritten, dass domänenspezifische Modelländerungen mit verhältnismäßig wenig Aufwand umgesetzt werden können. So verschlingt das Hinzufügen eines Attributs an eine fachliche Entität keinen Entwicklungsaufwand. Das Datenbank-Schema wird generiert, die Importer-Schnittstelle wird direkt erweitert, das grafische Modellierungswerkzeug Xplor bietet dem Benutzer die Modellierung des neuen Attributes an und ein Datenbank-Migrationsskript wird automatisch erzeugt. Einzig das Exporter-Modul, d. h. das oAW-Template, muss manuell angepasst werden, falls man mit dem neuen Attribut eine neue Exporter-Funktionalität erzeugen möchte. Diese Aspekte geschehen automatisch auf Knopfdruck. Natürlich bleibt einem der Aufwand rund um das Releasemanagement nicht erspart.

Quo vadis CEISeR?

Mit unserem strategischen Partner erarbeiten wir im Augenblick eine Evolutionsstrategie von CEISeR. Das Ziel der Evolution ist, dass unsere Anforderungen durch ein Standard-Produkt erfüllt sind und damit die Mehrwerte einer Produktsuite für uns nutzbar werden.

Links und Literatur

[oAW] openArchitectureWare, `http://www.openarchitectureware.org`
[SeKa08a] C. Sensler, A. Karalus, SOA @ T-Mobile – Change- und Configuration Management hoch 5 oder das Pentagon der T-Mobile, SubConf, 2008
[SeKa08b] C. Sensler, A. Karalus, SOA@T-Mobile – Vollautomatische Service Provisionierung auf dem ESB – Teil 1-3, in: Java Magazin, 10.2008 – 12.2008, `http://www.sensler.de/public.html`
[SeKa08c] C. Sensler, A. Karalus, SOA@T-Mobile – Vollautomatische Service Provisionierung, W-JAX, 2008
[Sun] Connection (Java Platform SE 6), Sun Microsystems. `http://java.sun.com/javase/6/docs/api/java/sql/Connection.html#TRANSACTION_SERIALIZABLE`

Dipl.-Ing. Carsten Sensler ist Angestellter der T-Mobile Deutschland GmbH und füllt dort die Rolle des Teamleiters eines internationalen Teams zur Service-Provisionierung aus. Zudem arbeitet er an den Konzepten der SOA-Backplane mit und gibt nationale sowie internationale Trainings bzgl. SOA-Backplane. E-Mail: publication@sensler.de.

Dipl.-Inform. Andre Karalus ist freiberuflicher Softwarearchitekt mit den Schwerpunkten Architektur, modellgetriebene Entwicklung und Integrationsframeworks. Seit seinem Abschluss an der RWTH Aachen 1996 war er als Consultant bei verschiedenen Kunden aus den Branchen Telekommunikation, Handel und Banken im Einsatz. Er arbeitet seit 2006 für die T-Mobile Deutschland GmbH; dort gestaltet er am Design der SOA-Backplane mit und unterstützt deren Umsetzung.

Markward Märtens leitet die Abteilung Enterprise Integration bei T- Mobile, die für die europaweite Delivery der T-Mobile-SOA-Infrastruktur zuständig ist. Seine Informatiklaufbahn startete er bei EDS und war als Berater in verschiedenen Projekten im Einsatz. Zur T-Mobile wechselte er 1998 und leitete u. a. die In-House-Entwicklung der CRM-Kernapplikationen. Seit gut vier Jahren beschäftigt er sich intensiv mit dem Thema SOA.