SOA@T-Mobile

Der erste Teil dieses zweiteiligen Artikels gibt eine Einführung in das SOA-Backplane-Programm von T-Mobile. In der nächsten Ausgabe des Java Magazins geht es dann um die vollautomatische Serviceprovisionierung auf dem ESB im Detail.

von Carsten Sensler und Andre Karalus

T-Mobile ist ein großer internationaler Konzern, in dem mittlerweile diverse Softwarekomponenten aus verschiedenen Landesgesellschaften miteinander kommunizieren müssen, um bestimmte Geschäftsprozesse für internationale Synergien (technischer und betriebswirtschaftlicher Art) zu unterstützen. Dies wird über einen internationalen ESB [1], der gleichzeitig eine SOA ermöglicht, geleistet. Das so genannte SOA-Backplane-Programm (SOA BP) stellt dazu die benötigte Infrastruktur bereit und definiert gleichzeitig einen Teil der technischen und organisatorischen Prozesse, die die Grundlage für die SOA Governance bilden. Das SOA-Backplane-Programm ist nicht nur ein Infrastrukturprogramm, sondern spiegelt auch die Ziele der IT-Strategie von T-Mobile wieder. Es sollen möglichst viele Komponenten international genutzt werden, um eine internationale Homogenität in der Anwendungslandschaft zu erzeugen und Betriebskosten zu senken. Langfristig sollen auch die Aufwände in jeder Landesgesellschaft reduziert und eine Vereinheitlichung der IT-Architektur und letztendlich auch der IT-Prozesse realisiert werden.

Bereits vor dem SOA-Hype gab es bei T-Mobile einen Vorläufer-ESB, der auch unter Nutzung von Web Services einen ubiquitären Zugriff auf Services bereitstellte. Dieser ist aufgrund einiger Unzulänglichkeiten im Begriff, abgelöst zu werden, und die damit gemachten Erfahrungen flossen ins SOA-BP-Programm ein.

Die Erfahrungen

- Das Service Repository (SR) bestand im Prinzip aus einem Rational-Rose-Modell, das nur einen bestimmten Entwicklungsprozess unterstützt, der nur in Deutschland etabliert war. Das SR ist nun eigenständig, datenbankbasiert, flexibler und unterstützt mehr Anwendungsfälle.
- Der alte ESB basierte direkt auf Tibco Rendezvous, wobei er die technische Schnittstelle ohne mandatorische Nutzung eines Gateways offenlegte. Der darauf basierende Message Exchange wurde von den fachlichen Komponenten immer wieder neu programmiert, was fehleranfällig war und keine vereinheitlichten Mittel zur Kontrolle des Nachrichtenaustauschs erlaubte, außer auf unterster Ebene, in den Rendezvous-Logdateien. In SOA BP erfolgt jeder Zugang zum ESB durch ein Gateway, den Common Access Layer (CAL), der Nachrichtenaustausch wird einheitlich auf höherer Protokollebene im Log- und Messagestore (LMS) nachvollziehbar gemacht.
- Bietet der ESB nur ein Notify/Observer Message Exchange Pattern (MEP) an, so sind etliche fachliche Komponenten für andere MEPs darauf angewiesen, auf Protokollebene eigenen Code zu entwickeln, was ebenfalls fehleranfällig und redundant ist. Das Gateway zum ESB unterstützt nun direkt die drei gebräuchlichen MEPs.

ESB und Gateways

Besonders in einem internationalen Unternehmen mit einer Vielzahl von Netzsegmenten, die zum Teil mit Firewalls voneinander abgeschottet sind, spielt der ESB seine Vorteile aus: Statt für jede neue (internationale) Konnektivität weitere Firewall-Freischaltungen zu erfordern, genügt es, einmalig den Bus zu installieren. Es hatte sich nämlich in früheren Projekten gezeigt, dass internationale Firewall-Freischaltungen sowohl aus technischer als auch organisatorischer Sicht ein höchst aufwändiger und fehleranfälliger Vorgang sind. Alle Komponenten benötigen dann nur Zugang zu ihrem nächsten Gateway, um potenziell mit einer beliebigen anderen Komponente, die an den Bus angeschlossen ist, zu kommunizieren. Dabei spielt es keine Rolle, ob es sich um eine Kommunikation innerhalb einer Landesgesellschaft handelt oder um eine Kommunikation zwischen zwei verschiedenen Landesgesellschaften.

Pro und Kontra statisches Routing

Das Routing legt fest, welcher Service Consumer (SC) mit welchem Service Provider (SP) verbunden ist, bzw. dass er dessen Service nutzen kann. Beim dynamischen Routing spezifiziert der SC im

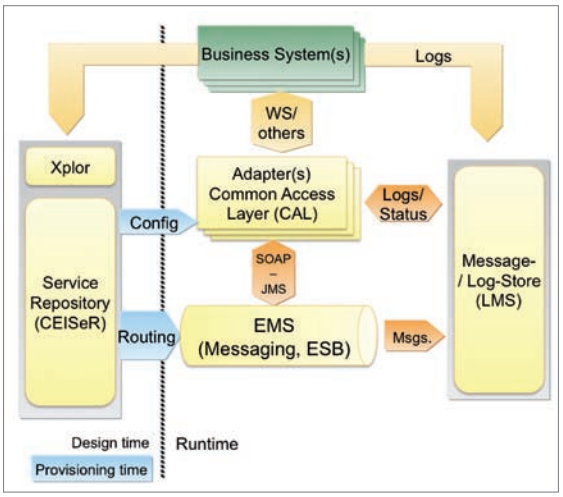

Abb. 1: Architekturübersicht SOA Backplane

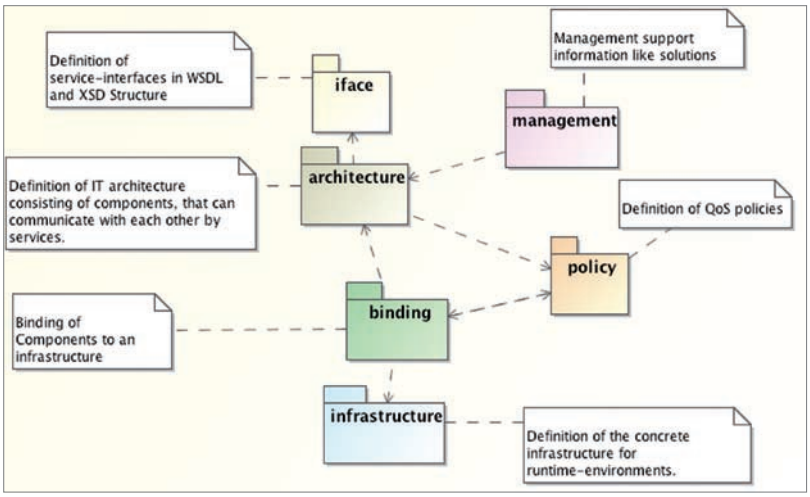

Abb. 2: Logische Schichtung des Service-Repositories

technischen Teil (z.B. ESB-spezifischer Header) der Nachricht, welchen SP er nutzen will. Der ESB routet die Nachricht dann aufgrund dieser Information adhoc. Beim statischen Routing spezifiziert der SC im technischen Teil der Nachricht, wer er ist und welchen Service er nutzen will. Der ESB ist so konfiguriert, dass er die Nachricht dann zum entsprechenden SP routet. Der Vorteil des dynamischen Routings – eine flexible Nutzung von Services zu ermöglichen – wird schnell wieder zunichte gemacht. In komplexen Umgebungen lässt sich bald nicht mehr erkennen, wer wen benutzt – eine Kontrolle ist nicht mehr möglich. So lässt sich z.B. nicht mehr sicher erkennen, ob ein Service Provider überhaupt noch genutzt wird, vielleicht noch über Protokollierung aller Nachrichten über einen aussagekräftigen Zeitraum.

SOA Governance macht es notwendig, dass Kommunikationsbeziehungen kontrolliert werden, dies unterstützt der Ansatz des statischen Routings am besten. Der Nachteil ist, dass bei Änderungen an den Routings der ESB neu konfiguriert werden muss.

Zur statischen Konfiguration des ESB werden in den Gateways Artefakte deployt, die Informationen über die möglichen Routings, Policies, Web Service Endpoints etc. enthalten. Dieser Vorgang ist ein Bestandteil der Serviceprovisionierung, die im zweiten Teil des Artikels genauer dargestellt wird. Zunächst soll aber das Zusammenspiel der SOA-BP-Infrastrukturkomponenten erläutert werden, damit der Gesamtkontext verständlich wird.

SOABP-Architektur

Als Erstes definiert das SOA-Backplane-Programm den ESB, die Laufzeitumgebung (Abb. 1).

Der SOA BP Core besteht aus einem Service-Repository (CEISeR), dem Common Access Layer (CAL) zum Zugriff auf den ESB und der Basisinfrastruktur zum Nachrichtenaustausch (ESB). Die Basisinfrastruktur verwendet den Messaging Standard JMS; als Produkt wird hierfür der Tibco-EMS-Server erfolgreich eingesetzt. Weitere Komponenten bilden die so genannten Added Values der SOA Backplane; es wurde eine zentrale Logging- und Monitoring-Komponente eingeführt (LMS) [2], die das Service-Contract-Management definiert, und Business Activity Monitoring steht kurz vor der Einführung. Als eine Art Regelwerk ist ein umfassendes Dokument erstellt worden, in dem aufgezeigt wird, wie innerhalb von T-Mobile die SOA verstanden und umgesetzt werden soll. Die SOA Governance ist nicht Bestandteil der SOA BP, sondern eine eigene Säule, sie wird durch die SOA BP unterstützt (z.B. statisches Routing).

Die Architektur besteht je nach Einsatzzeitpunkt aus drei Kategorien von Softwarekomponenten:

- *Designtime-Komponenten*: CEISeR (Service Repository), Xplor (Eclipse-RCP-Anwendung), CEISeR-Importer, Job-Execution-Manager
- *Provisioning Time*: Configstore, Local Service Provisioning Agents (LSPA), CEISeR-Exporter
- *Runtime*: ESB (Bestehend aus einem Verbund von EMS-Servern), Gateway und Zugriffsschicht zum ESB (CAL), Log-/Messagestore (LMS)

Wichtigstes Modellierungswerkzeug zur Designtime bei der SOA Backplane ist der Xplor; das zentrale Service Repository ist der Basis-Layer der Designtime-Komponenten. Abbildung 2 verdeutlicht die logische Schichtung innerhalb des Service-Repositories.

Diese sechs Ebenen wurden eingeführt, um die verschiedenen Arten von Informationen in dem Repository ablegen zu können. Gespeichert werden Serviceinterfacedefinitionen (*iface*) (WSDL), Architekturen (*architecture*) bestehend aus Applikationen, Komponenten und Ports, die als Service Provider oder Service Consumer modelliert werden, und konkrete physische Laufzeitumgebungen (*infrastructure*). Im Binding-Bereich (*binding*) werden die abstrakten Architekturen konkreten Laufzeitumgebungen (Entwicklung, Test oder Produktion) zugewiesen. In Abbildung 3 ist eine Zusammenfassung der Artefakte des Service Repositories aus der Im- bzw. Exporter-Sicht dargestellt.

Daraus resultiert eine umfassende statische Ansicht bezüglich einer spezifischen physischen Laufzeitumgebung: Welche Applikation bietet welche Services und welche Consumer sind für die jeweiligen Services angebunden? Aus Un-

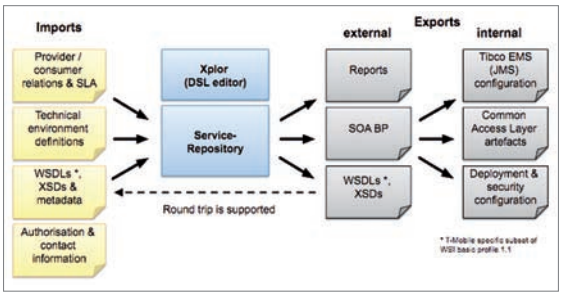

Abb. 3: CEISeR in a Nutshell

ternehmenssicht erhält man umfassende Informationen der Laufzeitumgebung, die bis dahin innerhalb von T-Mobile nicht derart konsolidiert vorhanden waren. Die Nutzung eines zentralen Service Repositories versetzt einen in die Lage, verschiedenartige KPIs leicht zu ermitteln (z.B. Grad der Wiederverwendung von Services/Datentypen). Für die Entwicklung des Service Repositories wurde ein modellgetriebener Ansatz verfolgt. Ausgehend von einem Metametamodell, das die Konzepte des Metamodells definiert, ist ein Metamodell für die SOA von T-Mobile entwickelt worden. Aus dem Metamodell werden durch den Einsatz des MDSD-Generator-Frameworks oAW [3] diverse Artefakte für die Persistenz- sowie für die Zugriffsschicht generiert. Für die Nutzung des Service Repositories ist die Jobmetapher umgesetzt worden. Es galt, bei der Konzeption des Service Repositories die verschiedenen Interessen der Integration bestehender Repositories bzw. Entwicklungsprozesse und Werkzeuge zu berücksichtigen. Hierbei liegt die Spanne zwischen einem bereits vorhanden Service Repository, das es zu integrieren galt, über vollautomatische Build-Ketten, in denen WSDLs generiert werden und die selbstverständlich automatisch in das Service Repository importiert werden können sollten, bis zu kleinen Projekten, die alle nötigen Artefakte händisch erstellen und importieren möchten. Aus diesem Grund haben wir uns als Zugriffsschicht des Service Repositories für eine Web-Service-Anbindung entschieden und unterhalb dieser Schicht einen Job-Execution-Manager bereitgestellt. Jeder Benutzer (entweder ein Buildscript, ein Cronjob oder ein realer Benutzer) stellt per Web Service einen Job in die Job-Queue ein und erhält synchron nach Ausführung des Jobs eine Rückmeldung über das Ergebnis. Gesteuert wird die Ausführung eines Jobs mit der so genannten Manifestdatei, die genau beschreibt, welche Tasks in dem Job abgearbeitet werden sollen. Abbildung 4 veranschaulicht den architektuellen Aufbau des Service Repositories.

Abbildung 5 zeigt das Modellierungswerkzeug Xplor. Es ist eine generische Anwendung, basierend auf der Eclipse-Rich-Client-Plattform, die zur Laufzeit das Metamodell interpretiert und das Layout daraus ableitet. Die DSL für die Modellierung der CEISeR-Input-Artefakte leitet sich direkt aus dem Metamodell ab. Technisch ist sie in einfachen XML-Konstrukten abgebildet. Die XML-Repräsentation (Abb. 6) wird allerdings durch die grafische Abstraktion des Xplor vor dem Benutzer gekapselt.

Serviceprovisionierung in a Nutshell

Eine Serviceprovisionierung ist das Freischalten der Kommunikation zwischen einem Serviceanbieter und Servicenutzer auf technischer Ebene. Hierzu werden alle nötigen Konfigurationen auf dem ESB und in den Gateways vorgenommen; vorzugsweise automatisiert.

Runtime

Der ESB der SOA Backplane besteht aus einer Zugriffsschicht und einem Gateway (CAL) als einzigem Zugriffspunkt auf den eigentlichen JMS-Bus. Der CAL läuft in einem JEE-Application-Server (JBoss) und enthält sämtliche Routing-Informationen für die fachlichen Komponenten, die den jeweiligen Zugriffspunkt benutzen. Für den Nachrichtenaustausch via JMS kommt der Tibco-EMS-Server zum Einsatz, und es wird eine Hub-and-Spoke-Architektur [4] eingesetzt. Die EMS-Server sind fault-tolerant aufgesetzt (hot standby) und die Applikationsserver mit CAL sind in mehreren Instanzen vorhanden – allerdings mit einem Loadbalancer vorgeschaltet, der für Lastverteilung und hohe Verfügbarkeit sorgt. Der Betrieb der CALs ist in der Produktionsumgebung unterbrechungsfrei, sowohl bei Rekonfiguration des Busses (durch die Serviceprovisionierung) als auch bei Softwareupdates der Zugriffsschicht (CAL).

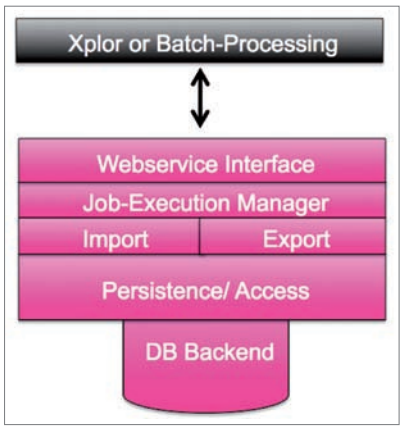

Abb. 4: Vereinfachte Architekturübersicht von CEISeR

Jede Landesgesellschaft stellt einen Spoke, der in ihrer eigenen Verantwortung liegt, und eine Gruppe von CALs – je nach Nachrichtenaufkommen zwischen zwei und fünf Instanzen. Mindestens zwei Instanzen und ein Loadbalancer müssen aufgebaut sein, um ein unterbrechungsfreies Update der Infrastruktursoftware sicherzustellen. Der Spoke jeder Landesgesellschaft verbindet sich über eine SSL-verschlüsselte Route zum Hub, dem zentralen EMS-Server. Im Konzept wurde berücksichtigt, dass ein Nachrichtenaustausch, der ausschließlich innerhalb einer Landesgesellschaft stattfindet, nur über den lokalen EMS-Server abgewickelt wird, um eine unnötige Latenz zu vermeiden und die Bandbreite der internationalen Leitungen zu schonen; nur ein internationaler Nachrichtenaustausch verläuft über den Hub.

Wie in Abbildung 8 dargestellt, erfüllt der CAL vielfältige Aufgaben [5]. Zunächst dient er im einfachen Fall als Web Service Gateway, d.h. er stellt einen Web-Service-Endpunkt zur Verfügung, über den die fachlichen Komponenten Zugriff auf den ESB via Web Service bekommen. Die SOAP-Nachricht wird dann über JMS geroutet (HTTP->JMS). Weiterhin implementiert er verschiedenartige MEPs:

- *Synchrones MEP (InOut)*: Als Gateway verschickt er die SOAP-Nachricht asynchron über den ESB (Request-Queue) und synchronisiert sie mit

der asynchronen Response (Response Queue), um für den Service Consumer ein synchrones MEP bereitzustellen. Dabei kümmert er sich um alle Fehlerbehandlungen, die bei der Umsetzung asynchron/synchron nötig sind, z.B. SOAP Fault bei Ausbleiben der asynchronen Response innerhalb eines (pro Nachricht) definierbaren Timeouts.

- *Asynchrones MEP (Robust In-Only)*: Die Nachricht wird über persistente Queues verschickt, der CAL auf der Seite des SP sorgt für eine verlässliche (reliable) Auslieferung, ist also die fachliche Komponente in der Rolle des SP nicht verfügbar, wird die Nachricht in definierbaren Intervallen wieder neu zugestellt, bis dies erfolgreich war. War dies in einem definierbaren Zeitraum nicht möglich, geht die Nachricht immer noch nicht verloren; in einer persistenten Undelivered Queue steht sie dann immer noch zur (manuell zu definierenden) Nachbearbeitung zur Verfügung. Die Response vom SP zu dieser Nachricht wird asynchron (als zweite *RobustIn-Only*-Nachricht) auf die gleiche Art und Weise zum SC ausgeliefert.
- *Notification MEP (Robust In-Only)*: Ähnlich wie beim asynchronen MEP wird hier aber die Nachricht intern über ein JMS-Topic verschickt. Es können im CEISeR mehrere Consumer dieser Notifizierung vorhanden sein, die dann zur Laufzeit über ihren CAL die Nachricht empfangen (Broadcast möglich). Außerdem können in CEISeR Filterkriterien pro Consumer definiert werden, die dafür sorgen, dass der Consumer nur die Nachrichten empfängt, an denen er interessiert ist.

Der CAL erzeugt für jeden Nachrichtenaustausch Logpoints. Dies sind ebenfalls JMS-Nachrichten, die von der LMS-Komponente in eine Datenbank geschrieben und von dort über ein GUI abgefragt werden können. Dies ist vor allem bei einem international gerouteten Nachrichtenaustausch nützlich, da dort der gesamte Nachrichtenaustausch von mehreren verteilten Laufzeitkomponenten realisiert wird, wobei der LMS eine Gesamtsicht auf die technischen Details des Nachrichtenflusses ermöglicht und damit in komplexen Problemfällen das Mittel zur Fehleranalyse ist.

Ein weiteres Feature des CALs sind Plug-ins, die es ermöglichen, Transformationen auf die Nachricht anzuwenden. Dies ermöglicht fachlichen Komponenten, die bereits eine technische Schnittstelle mithilfe einer anderen Middleware anbieten können, ohne größere Aufwände den Zugang zur SOA BP. Beispiele für Plug-ins sind:

- *Tuxedo Services*: Ein Plug-in, das generisch nach bestimmten Konventionen Tuxedo-Nachrichten in SOAP-Nachrichten mappt und umgekehrt.
- *Binary Mapper*: Fachliche Komponenten in der Rolle des SP, die eine beliebige *legacy*-Aufrufschnittstelle anbieten. Hier kann ein Java-Plug-in genutzt werden, das SOAP auf *byte[]* und umgekehrt abbildet. Dieses Plug-in muss für jeden Service neu erstellt werden, kann aber immerhin zusammen mit den Routing-Informationen des Service mit provisioniert werden.
- *XSLT-Mapper*: Beliebige XML-Nachrichten können mittels XSLT in SOAP-

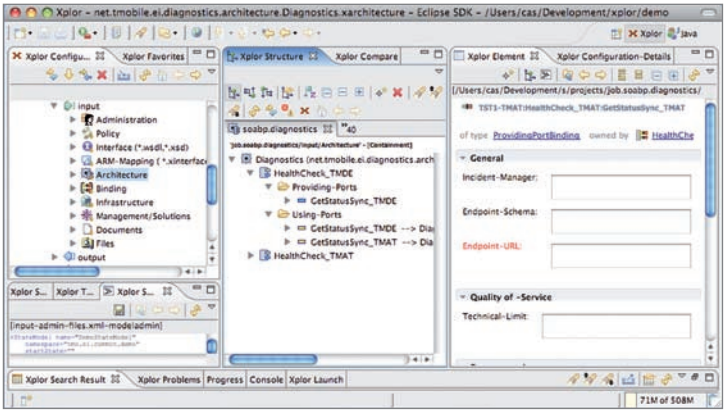

Abb. 5: Screenshot Xplor – Modell einer Provider- und Consumer-Applikation

Abb. 6: XML-Repräsentation des gleichen Modellausschnitts (Architektur) aus Abbildung 5

Kennzahlen und Performance

Es sind 110 internationale Services provisioniert. Insgesamt sind 700 Services provisioniert. Die Provisionierung eines nationalen Services dauert < 1 Minute, eines internationalen Services < 2 Minuten.

Abkürzungen

ESB – Enterprise Service Bus
CAL – Common Access Layer
CEISeR – Central Enterprise Intergation Service Repository
ESB – Enterprise Service Bus
GSPA – Global Service Provisioning Agent
KPI – Key Performance Indicator
LSPA – Local Service Provisioning Agent
MEP – Message Exchange Pattern
oAW – openArchitectureWare
SOA BP – SOA Backplane
SSL – Secure Socket Layer

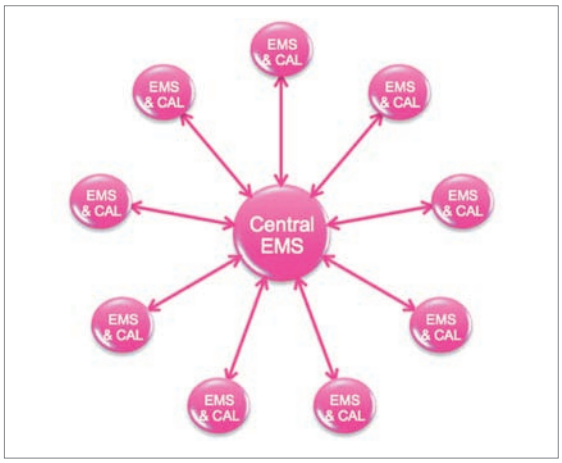

Abb. 7: Hub-and-Spoke-Architektur der SOA Backplane

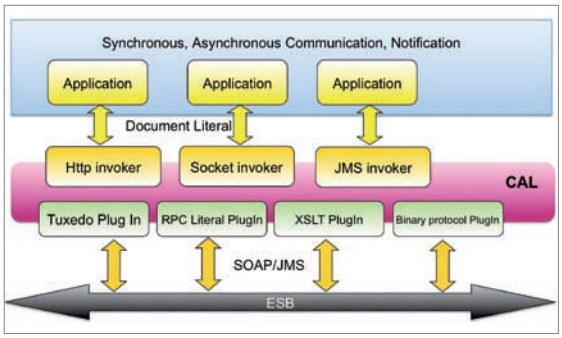

Abb. 8: Architekturübersicht Common Access Layer

Nachrichten transformiert werden und umgekehrt.

- *CORBA-Plug-in*: Stateless-Corba-Services können genutzt oder angeboten werden.

Der Vorteil der Plug-ins ist, dass sie im Prozess des CALs mitlaufen, dies senkt Latenzen und operationale Kosten. Der CAL definiert dafür „einen Container" für Plug-ins. Die Verteilung von Plug-ins erfolgt auch vollautomatisch über die Serviceprovisionierung. So werden z.B. die XSLTs zusammen mit den Routinginformationen verteilt. Dies ist ein weiterer Aspekt, die operationalen Kosten zu senken. Allerdings eignen sich nicht alle Schnittstellen von Legacy-Komponenten zur Anbindung durch Plug-ins. Es gibt folgende Einschränkungen:

- Sie müssen statusfrei sein (stateless).
- Ein SOAP-Aufruf muss eins zu eins auf einen Aufruf im Legacy-System mappbar sein (es ist keine Orchestrierung möglich).
- Die Semantik der Schnittstelle sollte nach Möglichkeit bereits dokumentenorientiert designt sein, denn feingranulare Aufrufe (z. B. *Kunde.getPLZ*) eignen sich nicht als Service für eine SOA-Integrationsplattform.

Nachrichtenformat

Innerhalb der SOA BP ist ein Nachrichtenformat definiert worden, das auf SOAP basiert. Allerdings werden weitere Einschränkungen gemacht, die der Interoperabilität dienen und das statische Routing sowie andere Features der SOA BP unterstützen. Die Definition des Nachrichtenformats geschieht durch WSDLs, die konform mit WSI Basic Profile 1.0 sein müssen und im Weiteren den *document literal wrapped Style* verwenden. Die WSDL-Message-Deklaration hat vor den fachlichen Payloads ein SOA-BP-spezifisches Element, den *eiMessageContext*. Dieser Header in der SOA-BP-Nachricht wird verwendet, um das statische Routing zu ermöglichen, indem der SC identifiziert wird. Der Standard WS-Adressing wurde nicht verwendet, da er zum Zeitpunkt der Einführung der SOABP nur von wenigen Web Service Stacks unterstützt wurde. Auf weitere Details des *eiMessageContext* werden wir im zweiten Teil eingehen.

Diese Anforderungen an die Nachrichten erfordern, dass selbst für fachliche Komponenten, die bereits eine Web-Service-Integration anbieten, in der Regel ein Plug-in erstellt werden muss (z.B. für SAP Netweaver).

Provisioning Time

Über den Web Service des Service-Repositories wird ein Job eingestellt, der aus dem SR eine Konfiguration für den ESB generiert. Diese wird dem *ConfigStore* per Web Service übergeben, der diese nach Überprüfung mittels des ESBs an die LSPAs der einzelnen Landesgesellschaften übermittelt. Die jeweilig zuständigen LSPAs provisionieren anschließend die Laufzeitkomponenten EMS-Server und CAL mit den Konfigurationen.

Fortsetzung folgt...

Der erste Teil umfasste die Grundlagen der SOA BP @ T-Mobile. Im zweiten Teil wird aufzeigt, was es mit der automatischen Serviceprovisionierung auf sich hat, welche Herausforderungen es im internationalen Kontext gab und wie bereits bei der Konzeption des Tool Chains der SOA BP (Xplor, CEISeR, GSPA, LSPA und dem CAL) auf die Anforderungen geachtet wurde. ■

Dipl.-Ing. Carsten Sensler ist Angestellter der T-Mobile Deutschland GmbH und ist dort Leiter eines internationalen Teams zur Serviceprovisionierung. Zudem arbeitet er an den Konzepten der SOA Backplane mit und gibt interne, internationale Trainings zu SOA Backplane.

Dipl.-Inform. Andre Karalus ist freiberuflicher Softwarearchitekt mit den Schwerpunkten Architektur, modellgetriebene Entwicklung und Integrations-Frameworks. Er arbeitet zurzeit für die T-Mobile Deutschland Gmbh. Dort gestaltet er am Design der SOA Backplane mit und unterstützt in der Umsetzung.

Links & Literatur

[1] Markus Demolsky: Enterprise Service Bus, in: Java Magazin 04.2008.

[2] Dirk Mögenburg: Service Monitoring in einer verteilten SOA-Umgebung, in: Java Magazin 06.2008.

[3] oAW: www.openarchitectureware.org

[4] Wikipedia: de.wikipedia.org/wiki/Hub_and_Spoke

[5] Carsten Sensler, Andre Karalus: Subconf 2007,Integration eines Service Repositories mit Subversion zur Anbindung an den ESB: 2007.subconf.de/fileadmin/PDF_Dateien/SubConf_2007/Vortraege/Integration_eines_SOA_Repositories_Sensler_Karalus.pdf

[6] Thomas Grimm, Michael Kunz: OOP 2007, SOA Backplane – MDSD-basierte SOA der T-Mobile.