SOA Referenzarchitektur

Klassifizierung *	**Nicht klassifiziert** / Intern / Vertraulich
Status **	**In Arbeit** / In Prüfung / Abgeschlossen
Projektname	SOA-Voraussetzungen - SOA-Vorgaben
Projektabkürzung	SOVO - Service-Verzeichnis
Projektnummer	7766
Projektleiter	willy.mueller@isb.admin.ch
Auftraggeber	peter.fischer@isb.admin.ch
Autor	willy.mueller@isb.admin.ch
Initiale	muw
Bearbeitende	muw
Prüfende	Wolfgang Tietz; Philipp Hoernes
Genehmigende	Projektausschuss
Verteiler	Projektmitarbeiter, Projektausschuss

* Nicht klassifiziert, Intern, Vertraulich
** In Arbeit, In Prüfung, Abgeschlossen

Änderungskontrolle, Prüfung, Genehmigung

Version	Datum	Beschreibung, Bemerkung	Name oder Rolle
1.0	2008-06-08	Erste Fassung aufbauend auf der Referenzarchitektur des EVD	Philipp Hoernes

Definitionen, Akronyme und Abkürzungen

siehe [7] für eine ausführliche Zusammenstellung

Begriff / Abkürzung	Bedeutung
SOVO	Projekt SOA-Voraussetzungen
SOA	Service Orientierte Architektur

Referenzen

Erkennungszeichen	Titel, Quelle
[1]	SOVO: SOA Governance Policies V1.0, ISB, in Erstellung
[2]	SOVO: SOA Governance Rollen und Prozesse V1.0, ISB, in Erstellung
[3]	SOVO: SOA Prinzipien V1.0., ISB, in Erstellung
[4]	SOVO: SOA Referenzarchitektur V1.0, ISB, in Erstellung
[5]	SOVO: SOA Best Practices V1.0, ISB, in Erstellung
[6]	SOVO: SOA Checklisten V1.0, ISB, in Erstellung
[7]	SOVO: SOA Glossar V1.0, ISB, in Erstellung
[8]	SOVO: Templates für Servicebeschreibungen und Kontrakte, ISB, in Erstellung
[9]	S007 - Teilstrategie SOA Bund 2008-2012, Willy Müller 2007
[10]	Weisungen des IRB über die Informatiksicherheit in der Bundesverwaltung vom 27. September 2004 (Stand 1. November 2007) , Anhang 2 und 3; Peter Grüter, 1.11.2007
[11]	SSZ-Domänen-Policy; Thierry Peraud, 20.3.2006

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Zweck des Dokuments

Das Strategieorgan für die Informatik der Bundesverwaltung (ISB) hat eine SOA-Strategie als Teilaspekt der übergreifenden IKT-Strategie festgelegt [9]. Darin wird das Ziel festgeschrieben, dass in der Bundesveraltung Services bereitgestellt werden sollen, auf deren Basis eGoverment Anwendungen aller Art realisiert werden können. Sowohl die Fachseiten als auch die Leistungserbringer sollten die fachlichen, technischen und organisatorischen Voraussetzungen für die Realisierung dieses Ziels schaffen.

Die in diesem Dokument vorgestellte Referenzarchitektur soll also nicht nur die grundlegenden Rahmenbedingungen für eine SOA-Realisierung festlegen[1], sondern auch die Voraussetzung für die konkrete Umsetzung der Pilotprojekte vorbereiten.
Im ersten Teil dieses Dokuments (Kapitel 2) werden die im Rahmen einer SOA potentiell relevanten Komponenten in ihrer Gesamtheit beschrieben. Darauf aufbauend werden in Kapitel 3 funktionale Patterns beschrieben, die diese Komponenten erfüllen müssen.

Die Referenzarchitektur definiert damit einem Rahmen der Anforderungen an technische Plattformen für Services aber auch an die Implementierungen von Service Providern und Service Consumern.

Im Sinne von TOGAF ist dieses Dokument primär als Artefakt der "Technology Architecture" (D) einzustufen.

Das Dokument richtet sich an IT-Architekten, Lösungsarchitekten und mit Einschränkungen auch an IT-Projektleiter.

[1] Und wird in diesem Zusammenhang auch als eine der Vorlagen für die Erstellung bundesweiter Vorgaben sein!

2 Komponenten der technische Referenzarchitektur

Im Rahmen der groben technischen Referenzarchitektur werden eine Reihe von technischen Komponenten definiert und deren Funktionalität und deren grundlegende funktionale Einsatzmuster (Patterns) beschrieben. Die Relationen dieser Komponenten untereinander werden dabei durch konkrete Patterns dargestellt.
Es handelt sich bei den hier beschriebenen technischen Komponenten um idealisierte Bausteine, die reale Produkte abtrahieren. Im Gegensatz zu realen Produkten sind nämlich die Funktionen der Komponenten der groben technischen Referenzarchitektur klar voneinander abgegrenzt.
Die grobe technische Referenzarchitektur hat den Anspruch, auf der Ebene der idealisierten Komponenten vollständig zu sein, also alle Aspekte von SOA abzubilden und auf alle Arten von SOA-konformer Software zu passen[2].
Dafür werden aber konkrete Anforderungen und Vorgaben aus der Bundesverwaltung nicht im Detail diskutiert, um eine höhere Allgemeingültigkeit zu erreichen. Es werden ggf. also auch solche Komponenten dargestellt, die im Bund auf absehbare Zeit keine Bedeutung haben werden.
Verfügbare technische Standards werden in der groben technischen Referenzarchitekur nicht eingehend beschrieben. Es findet auch keine Darstellung von Infrastruktur-Aspekten (Netzwerk, Betriebssysteme, Hardware, etc.) statt.

2.1 Schichten- und Komponentenübersicht

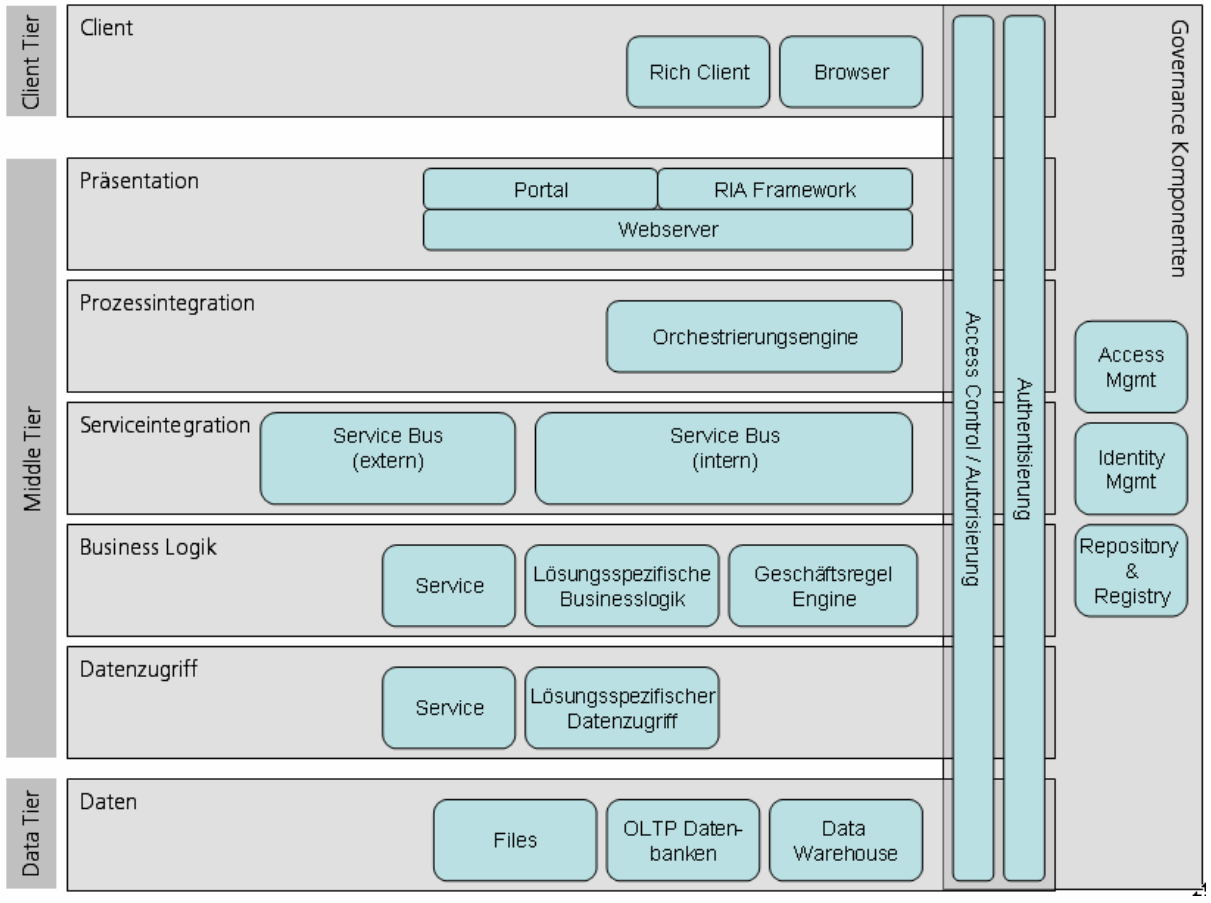

Abbildung 1: Übersicht über die Komponenten der groben technische Referenzarchitektur

2.2 Komponenten zur Datenhaltung

In dieser Schicht werden alle technischen Komponenten zusammengefasst, die zur Speicherung von Daten dienen. Darunter fallen aller Arten von operativen, transaktionsorientierten (OLTP) Datenbanken, einfache Dateien und datenbasierte Analysesysteme (Data Warehouses), die redundant Daten aus operativen Datenbanken halten und für Analysezwecke strukturieren.

[2] Wobei natürlich das heute Wissen und der aktuelle Stand der Technologie berücksichtigt wurden.

In dieser, auf SOA fokussierte Referenzarchitektur werden an diese Komponenten keine speziellen Anforderungen gestellt; vielmehr werden die detaillierten Anforderungen an Komponenten zum Datenzugriff an den bestehenden Vorgaben für Technologien im Bereich der Datenhaltung orientiert sein.

2.3 Komponenten zum Datenzugriff

Der Zugriff auf Daten erfolgt immer gekapselt, also niemals direkt aus den Komponenten heraus, die die Businesslogik abbilden. Je nachdem, ob ein bestimmter Datenzugriff potentiell wiederverwendbar ist und innerhalb von Services benötigt wird, erfolgt dieser durch Adapter Services. Danaben sind aber auch lösungsspezifische Komponenten zur Kapselung des Datenzugriffs möglich.

2.3.1 Services für den Datenzugriff

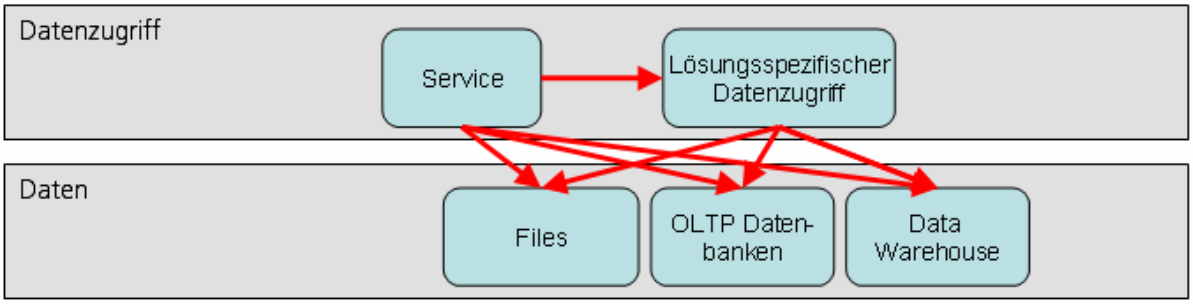

Abbildung 2: Zugriff auf Daten durch Adapter Services bzw. lösungsspezifische Komponenten

Definition: Ein Service für den Datenzugriff (oft auch „Adapter Service“ bzw. „Daten Service“ genannt) abstrahiert den Zugriff auf in einer Komponten der Datenhaltungs-Schicht gespeicherte Daten über ein wohldefiniertes und publiziertes Interface.
Services können auf den Layers „Datenzugriff“ und „Businesslogik“ auftreten. Typischerweise liegt das Hauptinteresse auf Services, die Business Logik abbilden – aber auch einfache Services für Datenzugriff können in vielen Fällen geschäftlichen Wert besitzen.

Funktionalität: Über das Interface eines solchen Services werden die benötigten Daten in einem definierten Format gelesen oder geschrieben.

2.3.2 Lösungsspezifischer Datenzugriff

Definition: Die innerhalb einer IT-Lösung realisierte Logik zum Zugriff auf Daten. Die Interfaces zum lösungsspezifischen Datenzugriff sind nur innerhalb des Kontexts der IT-Lösung bekannt und werden nicht darüber hinaus publiziert.

Funktionalität: Über das Interface einer Komponenten zum lösungsspezifischen Datenzugriff werden die benötigten Daten in einem definierten Format gelesen oder geschrieben.

2.4 Komponenten zur Abbildung von Businesslogik

Eine Grundidee der SOA ist die Abstraktion von Businesslogik, daher wird in der Architektur eine separate Schicht ausgeprägt, in der diese realisiert wird. Businesslogik wird in der Regel in Code (z.B. Java, C#, ...) realisiert. Es gibt allerdings auch die alternative Möglichkeit zur Nutzung von sog. Geschäftsregel-Engines (BRMS) zur deklarativen Abbildung von geschäftlich relevanten Regeln.
Wiederverwendbare Businesslogik kann – wie auch Logik zum Datenzugriff – als Service dargestellt werden.

2.4.1 Service (zur Abbildung von Businesslogik)

Definition: Ein Service abstrahiert Businesslogik, welche über ein wohldefiniertes und publiziertes elektronisches Interface abgerufen werden kann. Die funktionalen und nichtfunktionalen Eingenschaften eines solchen Service sind ebenso wohldefiniert und publiziert.

Funktionalität: Ein Service kann jedigliche Businesslogik darstellen, die über ein nachrichtenorientiertes Interface genutzt werden kann.

2.4.2 Lösungsspezifische Businesslogik

Definition: Jedigliche Businesslogik, die nur innerhalb einer IT-Lösung von Interesse ist (man spricht auch von "privater" Businesslogik).

Funktionalität: keine nähere Spezifikation, jedigliche Art von "hart codierter" Businesslogik

2.4.3 Geschäftsregeln Engine (BRE)

Definition: Eine Geschäftsregel Engine, engl. Business Rule Engine ist eine Umgebung, in der Geschäftsregeln definiert, verwaltet und ausgeführt werden können. Die Idee ist es dabei, komplexe bzw. variable Regeln nicht hart zu kodieren, sondern separat auf einer spezialisierten Plattform abzubilden.

Funktionalität: Eine Business Rule Engine ist eine Variante zum Implementierung von Businesslogik. Sie erlaubt die Definition von Geschäftsregeln auf eine Art und Weise (z.B. grafisch, oder in natürlicher Sprache), die von fachlichen Spezialisten beherrscht werden kann. Die Plattform muss die Verwaltung eines Vielzahl von Regeln inkl. deren Versionierung erlauben. Regeln müssen getesten und ausgeführt werden können.

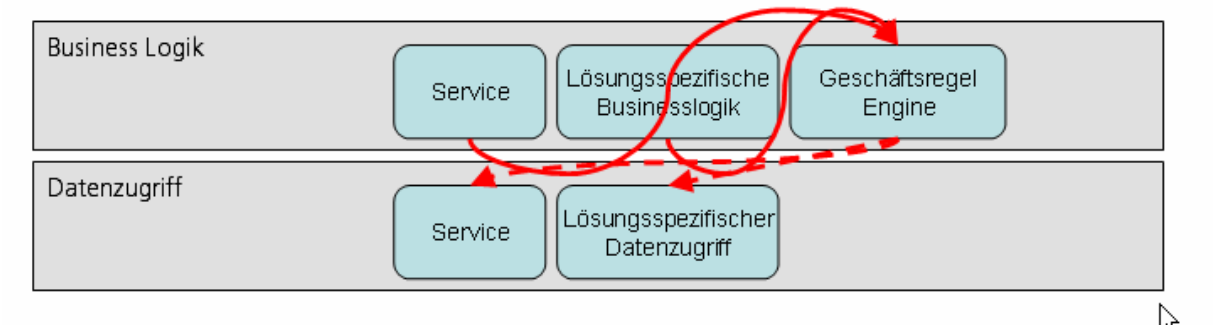

Abbildung 3: Eine Business Rule Engine implementiert Businesslogik und stellt diese Services oder privaten Komponenten zur Verfügung

2.5 Komponenten zur Serviceintegration

Das Thema Serviceintegration steht im Zentrum von SOA, und entsprechend die die in diesem Abschnitt beschriebenen Komponenten von grösster Bedeutung. Unter „Serviceintegration“ versteht man generell die Koordination der Funktionalitäten aller Services im Gesamtsystem. Konkret wird unterschieden zwischen

1. der internen Integration von Services innerhalb des Scopes der SOA (typischerweise innerhalb einer Organisation oder eines Unternehmens; hier: innerhalb des EVD) und

2. die „externe Integration“, mit der die Nutzung von ausserhalb des EVD angebotener Services und die externe Nutzung von durch das EVD angebotener Services ermöglicht wird.

Wichtig ist, dass es durchaus mehrere physikalische Ausprägungen der logischen Komponente „Service Bus“ geben kann. So können spezialisierte Busse einmal die externe und einmal die interne Integration übernehmen; aber auch mehrere interne Busse für verschiedene Aufgaben sind möglich.
Produkte, die die hier unter „Service Bus“ beschriebenen funktionalen Patterns realisieren, werden meist als Enterprise Service Busse (ESB) angeboten. Verschiedene ESB-Produkte unterscheiden sich in der Regel durch ihre nichtfunktionalen Eigenschaften oder dadruch, das diese bestimme Patterns mehr oder weniger effizient unterstützen. Im Rahmen der detaillierten Referenzarchitektur erfolgt eine Auswahl anhand der gewichteten funktionalen und nichtfunktionalen Anforderungen aus Projekten, die die SOA Infrastruktur des ISCeco verwenden wollen.

2.5.1 Service Bus (Allgemein)

Die Produktkategorie "Enterprise Service Bus" ist ungenau abgegrenzt und wird für eine Vielzahl unterschiedlicher Technologien und ein sehr breites Spektrum von Funktionen benutzt. Hier werden daher die wesentlichen Funktionalitäten eines abstrakten "Service Bus" unabhängig von konkreten Produkten und Technologien diskutiert.

Definition: Ein Service Bus ist eine Middleware, über die die gesamte Kommunikation mit Services in einer SOA abgewickelt wird. Der Service Bus entkoppelt die Bereitstellung von Services von deren Nutzung. Der Service Bus ermöglicht die flexible Umsetzung nichtfunktionaler Anforderungen (insbesondere an Sicherheit) durch Konfiguration.

Bedeutung und Relationen mit anderen Komponenten: Durch die Entkopplung der Service-Bereitstellung von der Service-Nutzung wird der Service Bus zur zentralen Komponente, über die die gesamte für die Integration wesentliche Kommunikation abgewickelt wird. Man kann auch sagen, dass der Service Bus als „Proxy“ auftritt, über den alle Services angeboten werden.

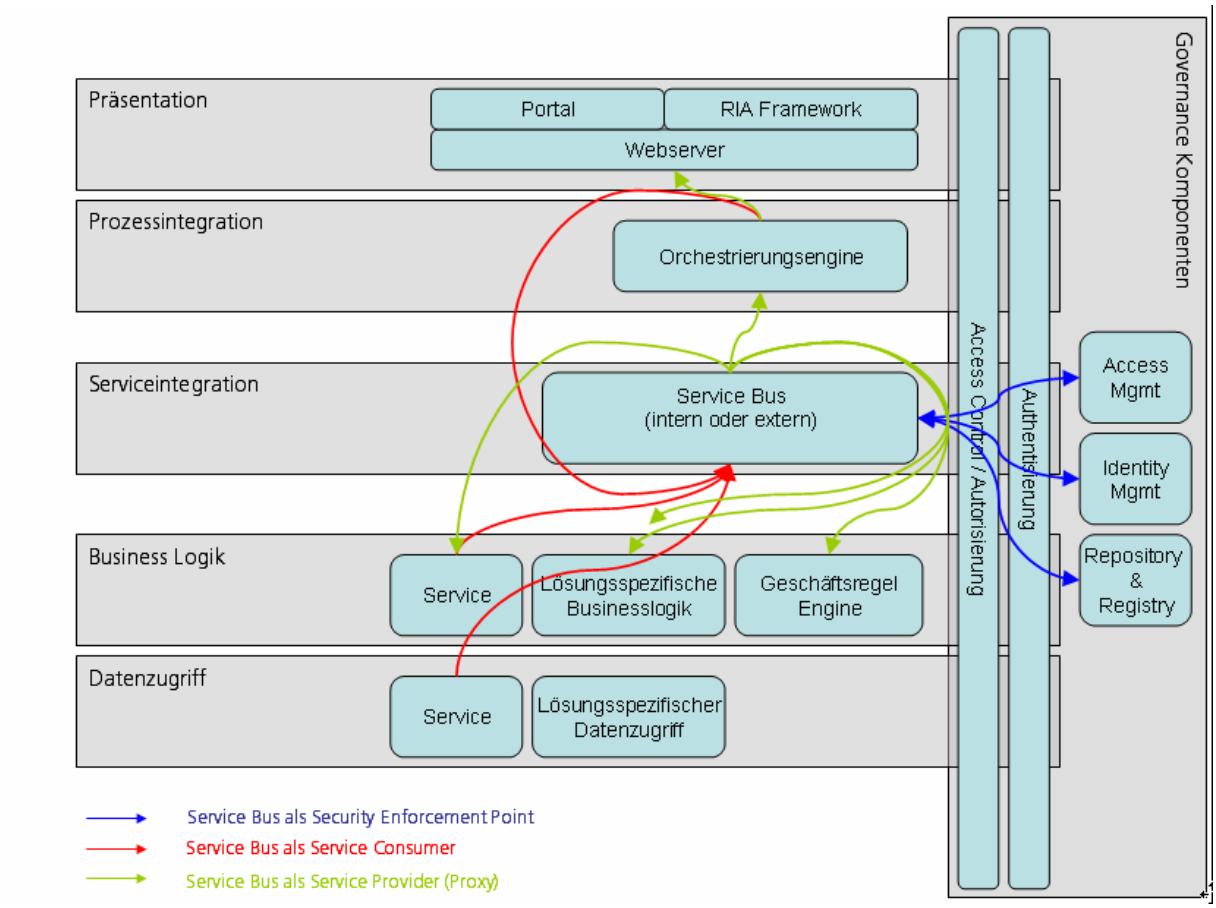

Abbildung 4: Der Service Bus in der Rolle als Provider / Proxy, als Consumer und als Security-Enforcement Point

Wichtig ist auch, dass Services so über den Service Bus auch von Komponenten auf „niedrigeren“ Schichten konsumiert werden können. So können Services, die an beliebiger Stelle und auf beliebige Art realisiert werden, in jeder Komponente der Business Logik konsumiert werden. Das gilt auch für „extern“ realisierte Services, und solche, die durch Prozessintegration in der Orchestrierungsengine realisiert werden und damit einen Businessprozess repräsentieren.

Ein zentrales Muster beim Einsatz eines Service Bus ist - neben der Funktion als „Proxy“ bzw. als Erweiterung dieses Konzeptes - die Trennung von Businesslogik und nichtfunktionalen Anforderungen. Die Idee ist es, die Implementierung von Business Services weitgehend unabhängig von nichtfunktionalen Anforderungen zu halten und letztere konfigurativ im Service Bus umzusetzen[3].

[3] Dies ist allerdings nicht immer voll umsetzbar.

Im Rahmen der Umsetzung einer fachlich getriebenen SOA ist der Service Bus wichtig, um verschiedene Business Domänen logisch voneinander zu trennen. So sollen sich (siehe [1]) die Domänen gegenseitig übergreifend definierte Business Services zur Verfügung stellen, über die die gesamte Inter-Domänen Kommunikation abgewickelt werden soll. Der Service Bus kann dazu auch von einzelnen IT Lösungen bereitgestellte einfachere (oder Domänenspezifische) Services zu Business Services zusammensetzen („composition“).

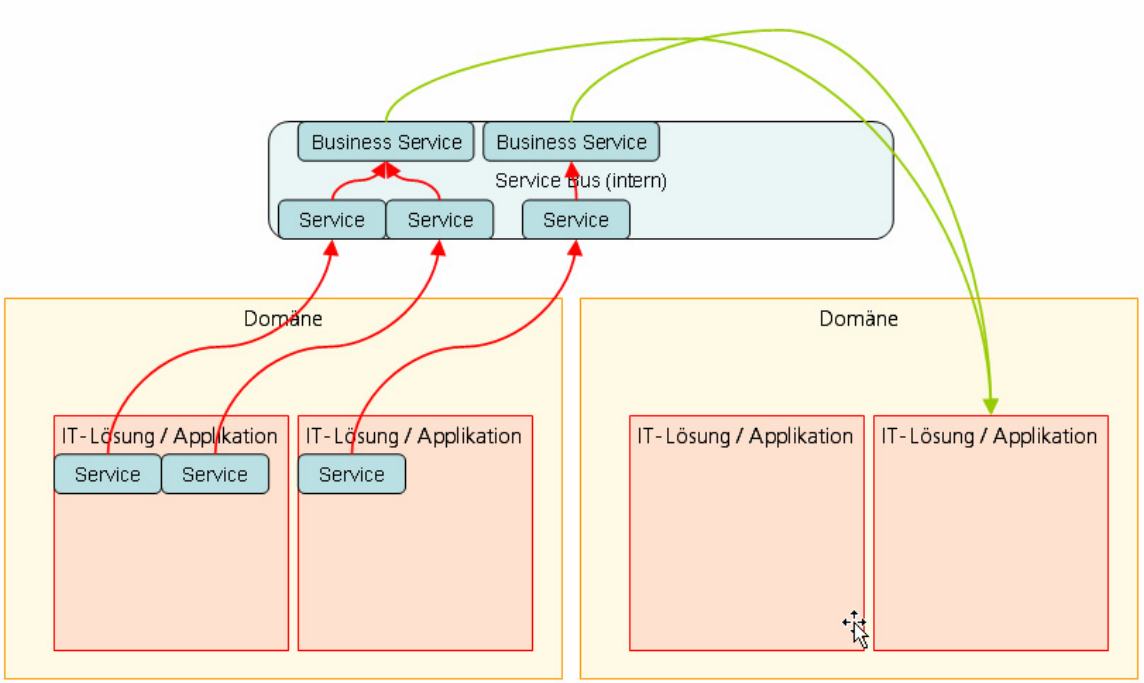

Abbildung 5: Service Bus zur Komposition von Services und zur Trennung von fachlichen Domänen (Business Domains)

Funktionalität: Ein Service Bus dient also dazu, Services unanhängig von deren Implementierung übergreifend zur Verfügung zu stellen. Die dazu notwendigen Funktionalitäten sind im wesentlichen Konnektivität mit verschiedenen Protokollen, die (auch komplexe) Verarbeitung von Messages, und die Überwachung. Ausserdem kann ein Service Bus auch die Umsetzung sicherheitsrelavanter Regeln garantieren.

2.5.2 Service Bus (Extern vs. Intern)

Sowohl ein “interner” als auch ein “externer” Service Bus erfüllen auf einer groben Ebene prinzipiell dieselben funktionalen Patterns. Sowohl deren Umsetzung im Detail als auch vor allem die nichtfunktionalen Eigenschaften werden allerdings je nach Einsatzgebeit voneinander abweichen.
Typischerweise kann ein „externer“ Service Bus vor allem Sicherheitsaspekte gut umsetzen; oft liegt der Fokus auf den Patterns für asynchrone Kommunikation. Die Integration mit anderen Komponenten (wie etwa einer Orchestrierungsengine, Registry, etc.) ist dagegen weniger wichtig.
Ein „interner“ Service Bus muss dagegen sowohl synchrone als auch asynchrone Kommunikation beherrschen. Die Performance der Messageverabeitung spielt – vor allem im synchronen Fall – eine sehr grosse Rolle. Sicherheitsfragen können dagegen weniger bedeutend sein.

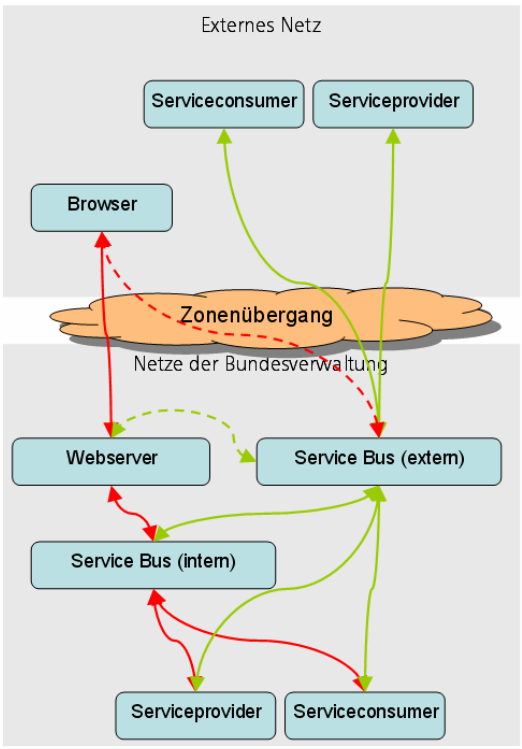

Abbildung 6: Schema der Nutzung von externem und internem Service Bus

2.6 Komponenten zur Prozessintegration

2.6.1 Orchestrierungsengine

Definition: Eine Orchestrierungsengine ermöglicht die Abbildung von Businessprozessen durch den koordinierten Aufruf von Businesservices ("Orchestrierung"). Der Businessprozess wird dabei durch ein Modell formal dargestellt und definiert, welche Businessservices in welcher Reihenfolge unter welchen Bedingungen aufgerufen werden. Das Ergbnis einer solchen Orchestrierung kann wiederum als Service exponiert werden.

Funktionalität: Eine Orchestrierungsengine muss die Definition, Verwaltung und die Ausführung von Businessprozessen ermöglichen. Der Status von Businessprozessen muss dabei von der Engine verwaltet und persistiert werden, so das insbesondere langlaufende Prozesse ermöglicht werden. Die Reaktion auf Fehler muss inneralb der Definition von Businessprozessen definiert werden können ("exception handling"). Die Überwachung und Protokollierung des Ablaufs von Businessprozessen muss möglich sein, ebenso der manuelle Eingriff im Falle von (schweren) Fehlern. Die Definition von Businessprozessen muss rekursiv sein dürfen: Ein gesamter Prozess muss als Service exponiert werde und wiederum in anderen Businessprozessen konsumiert werden können.
.

2.7 Komponenten zur Präsentation

Die Schicht „Präsentation“ ist für die Server-gestützte Interaktion mit dem Benutzer unter Nutzung grafischer Benutzeroberflächen zuständig. Im weitesten Sinne sind damit „Webanwendungen“ gemeint. Klassische Client-Server Anwendungen nutzen im Gegensatz dazu keine Komponente auf der Präsentationsschicht, da das ganze Handling der Nutzeroberflächen dort auf dem Client liegt und nur Businesslogik vom Server (vom Backend) angeboten wird.
Da die vom Markt angebotenen Produktkategorien teilweise überlappen, definieren wir hier drei klar abgegrenze logische Komponenten, nämlich „Webserver“ für reine Webapplikationen, zusätzlich (optional) ein „RIA Framework“ für erweiterte („reiche“) Webapplikationen und ein „Portal“ für die koordinierte Nutzung mehrerer Webapplikationen durch einen Nutzer. Die genaue Art der Zusammenarbeit dieser drei Komponenten hängt im Detail von den ausgewählten Produkten ab.

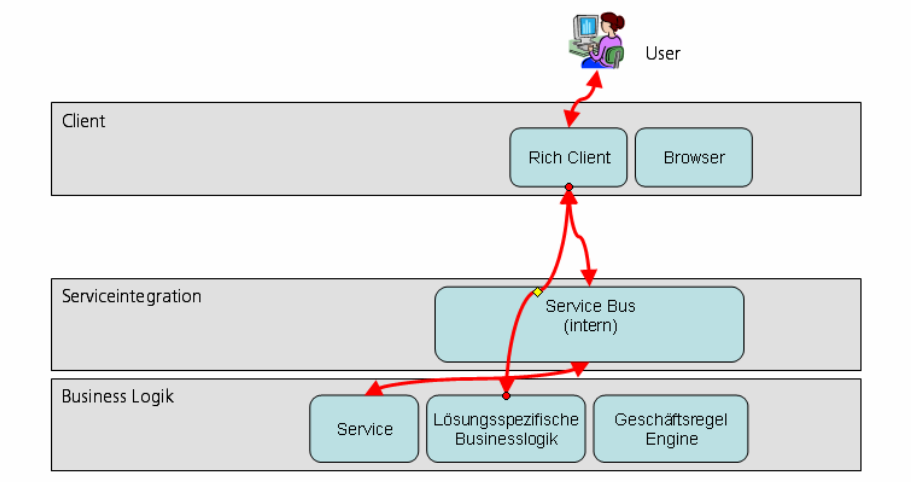

Abbildung 7: Schema der Interaktion von Komponenten bei einem „Rich Client“

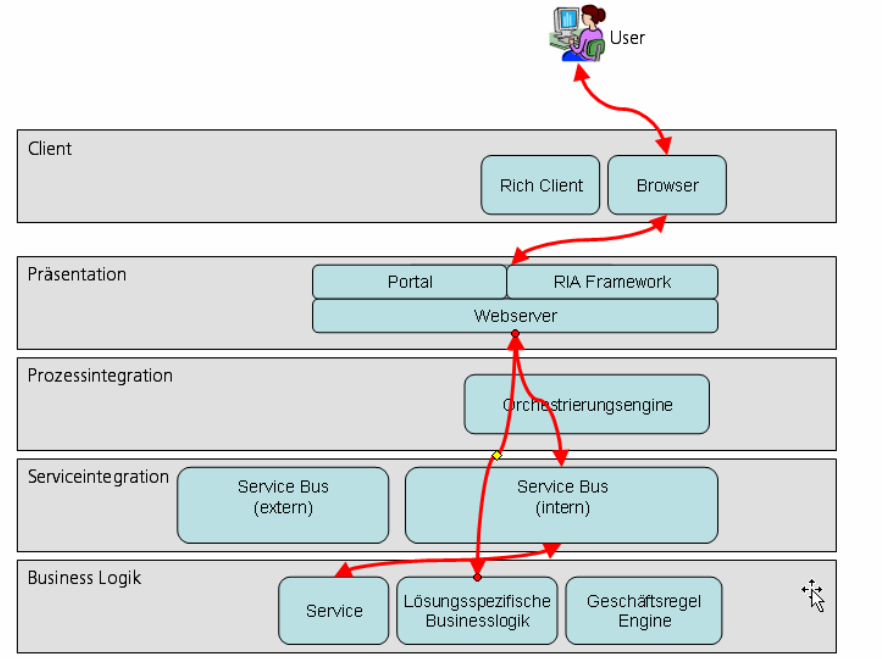

Abbildung 8: Schema der Interaktion der Komponenten bei Webapplikationen

2.7.1 Webserver

Definition: Ein Webserver ist eine Plattform, auf der Webapplikationen (also Applikationen, die clientseitig über einen Brower genutzt werden können) ablaufen[4].

Funktionalität: Ein Webserver ist eine Laufzeitumgebung für Webapplikationen. Die hauptsächliche Funktionalität ist die Bereitstellung (das "Rendering") von HTML-Seiten, die Entgegennahme und Verifizierung von Usereingaben und der Aufruf von Komponenten, die Geschäftslogik abbilden.

2.7.2 RIA Framework

Definition: Ein RIA (Rich Internet Application) Framework dient zur Realisierung von Webappliaktionen mit (gegenüber klassischen HTML-basierten) erweiterten Möglichkeiten zur Gestaltung des Userinterfaces.

[4] Sog. Applikationsserver schliessen einen Webserver ein

Funktionalität: Die erweiterte Funktionalität von RIA besteht in der Möglichkeit zur asynchronen Kommunikation mit dem Client (z.B. während Usereingaben) und der dynamischen und reichen graphischen Gestaltung der Oberfläche.

2.7.3 Portal

Definition: Unter einem Portal versteht man eine Komponente, die den Zugang zu mehreren Webapplikationen regelt und speziell die gleichzeitige Nutzung mehrerer Webapplikationen erlaubt.

Funktionalität: Portale bieten eine einheitliche Zugangskontrolle (login), erlauben die Auswahl von Webapplikationen, die gleichzeitige Nutzung mehrerer Webapplikationen auf einer Browserseite und ggf. auch die Möglichkeit zum Datenaustausch zwischen diesen.

2.8 Komponenten auf dem Client

2.8.1 Browser

Definition: Ein Browser ist ein Software zum Abruf von Seiten im World Wide Web (WWW).

Funktionalität: Rendering der HTLML-Seiten im Fenster, Kommunikation über http/https mit den Webservern.

2.8.2 Rich Client

Definition: Ein Rich Client ist eine Applikation, die lokal auf dem Client läuft und sowohl die Bereitstellung der Benutzeroberfläche als auch die Kommunikation mit der Businesslogik auf dem Server (dem "Backend") übernimmt.

Funktionalität: Darstellung einer Benutzeroberfläche, Verarbeitung der Benutzereingaben, lokale Businesslogik auch in grösserem Ausmass, Kommunikation mit dem Backend. Auch lokale Persistierung ist möglich und dann sinnvoll, wenn der Client nicht immer mit dem Backend Verbindung aufnehmen kann.

2.8.3 Nicht im Scope der Referenzarchitektur: Lokale Applikationen

Definition: Jede auf dem Client laufende Software, die nicht mit Businesslogik auf dem Server kommuniziert, ist eine lokale Applikation.

Funktionalität: Lokale Applikationen bieten jede denkbare Art von Funktionalität, allen voran sind Office-Programmpakete (etwa MS Office oder OpenOffice) zu nennen.

2.9 Governance Komponenten

2.9.1 Authentisierung

Definition: Unter Authentisierung versteht man die Identifikation von individuellen Nutzern, also zur Gewinnung von Identitäten aus den eingegebenen User Credentials.

Funktionalität: Bei der Authentisierung werden User Credentials (z.B.: Username / Passwor, Fingerabdrücker etc.) überprüft und einem Nutzer zugeordnet, also eine Identität zugeordnet (identifiziert). Dies kann und muss an verschiedenen Stellen geschehen (Client, Service Bus, Datenbank, ...) . Wichtig ist die sichere Weiterleitung von Identitäten, um mehrfache Authentisierungen zu vermeiden.

2.10 Autorisierung / Access Control

Definition: Unter Autorisierung versteht man die Gewährung von Rechten zur Nutzung von Software-Funktionalität an authentisierte (d.h. eindeutig identifizierte) Nutzer.

Funktionalität: Bei der Autorisierung geht es darum, identifizierten Nutzern (mit bekannter Identität) selektiv Zugang zu Funktionalitäten von IT-Systemen zu gewähren. Oft verwendet man man User, Usergruppen, Rollen auf User- oder Usergruppenebene und die Zuordnung von Rollen zu Rechten. Rechte können etwa durch Aktionen und Ressourcen ausgedrückt werden, so kann etwa eine bestimmte Rolle das

Recht erhalten, einen Service zu nutzen oder etwa einen bestimmten Datensatz zu ändern. Die Vergabe von Rechten und die Definition von Rollen wird durch die Komponente „Access Management“ geleistet.

2.10.1 Identity Management

Definition: Unter Identity Management versteht man die Verwaltung der Gesamtheit der Identitäten von Nutzern.

Funktionalität: Das Identity Management kann die Identitäten der Nutzer (zentral bzw. auch dezentral) anlegen, modifizieren bzw. entfernen.

2.10.2 Access Management

Definition: Das Access Management verwaltet und pflegt die Rechte und deren Zuordnung zu Nutzern.

Funktionalität: Die Vergabe von Rechten und die Definition von Rollen ist eine fachliche und administrative Aufgabe, die durch die Komponente Access Management unterstützt wird.

2.11 Service Repository und Registry

Definition: Sowohl das Repository als auch eine Registry speichern zentral Informationen über Services und dienen so als Referenz für die Publikation von Service-bezogenen Daten. Beide können also als "Verzeichnis" betrachtet werden.
Eine Registry dient vorrangig zum Abruf von technischen Informationen zur Laufzeit, demgegenüber speichert das Repository Dokumente aller Art, die für das Service Design ("zur Design-Zeit") von Bedeutung sind.

Funktionalität: Zu den Funktionen einer Registry gehört vorallen der "Lookup" von Services aufgrund deren eindeutiger Bezeichnung und die Rückgabe von zur Laufzeit relevanten Dokumenten (WSDL, XSD, ggf. Policies).
Ein Repository kann Dokumente aller Art speichern und verwalten sowie vor allem Abhängigkeiten zwischen verschiedenen Artefakten abbilden und so Inkonsistenzen vermeiden bzw. aufdecken.
Die Registry-Funktionalität kann auch ein Feature eines Repository-Produktes sein.

2.12 Spezialfälle und deren Abbildung in der Referenzarchitektur

2.12.1 Integrierte ("Legacy") Applikationen

Um der Tatsache Rechnung zu tragen, dass SOA nicht "auf der grünen Wiese" aufgebaut werden kann, müssen in der Referenzarchitektur sogenannte "Legacy Applikationen" bzw. integrierte Applikationen betrachtet werden. Das sind die, die sich nicht an die Referenzarchitektur halten, also entweder in Bezug auf die Technologie und / oder die eigentliche Architektur abweichen. Typischerweise bringen integrierte Applikationen (z.B. die Suiten von SAP, Siebel, etc.) ihre eigenen Server- (Businesslogik und Datenhaltung) und Client-Komponenten mit und die interne Kommunikation erfolgt auf proprietäre und nicht transparente Art und Weise, wobei meist keine echte Integrationsschicht implementiert wird.
Die Integration von solchen Legacy Applikationen in eine SOA kann von grosser Bedeutung sein und definiert entscheidende Anforderungen an die Produkte, insbesondere etwa die Connectivity-Fähigkeiten eines Service Bus.

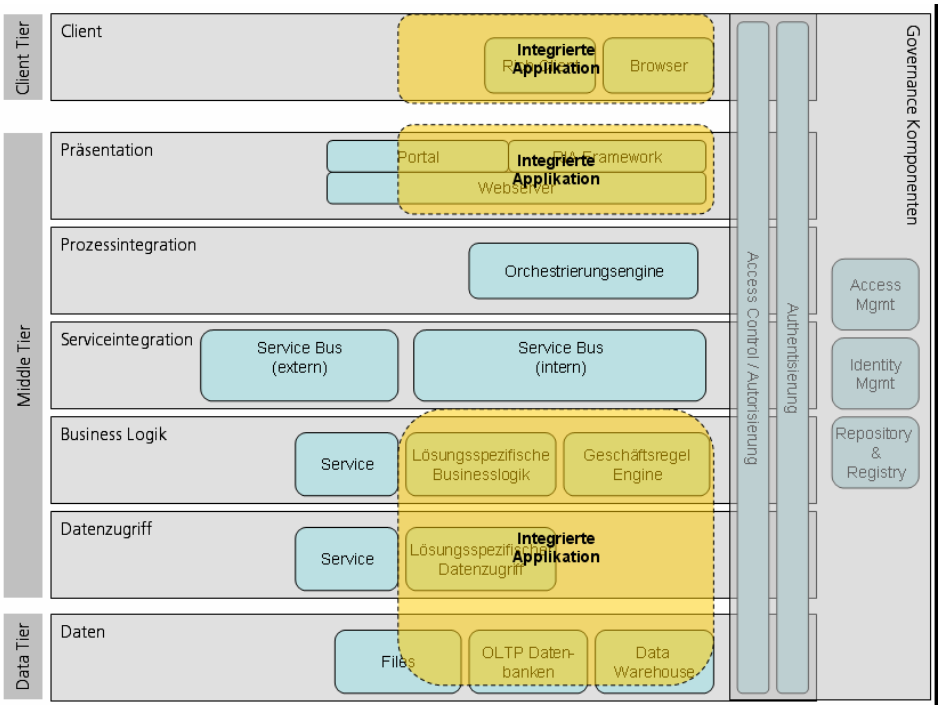

Abbildung 9: Integrierte „Silo“ Applikationen in der Referenzarchitektur

In den Abschnitten über "Adapter Services" und "Business Services" (siehe 2.3.1und 2.4.1) wird die Anbindung von Legacy Systemen in die SOA-Welt diskutiert.

2.12.2 Business Intelligence Lösungen

Definition: Unter Business Intelligence (BI) versteht man jede Art der organisierten Sammlung und Analyse von Geschäftsdaten in elektronischer Form.
In der Regel werden BI Lösungen in Zusammenhang mit Technologien wie Datawarehouses (DWH), aber auch mit Business Activity Management gesehen.
BI Lösungen und SOA stehen nicht im Widerspruch zueinander, auch da der Fokus ein völlig anderer ist. In der hier diskutierten groben Refernenzarchitektur würden BI Lösungen vorrangig auf der Daten-Schicht abgebildet werden: Die Population von Datawarehouses ist ein reines Datenthema und repräsentiert keine Businesslogik, muss daher auch nicht in Form von Services abgebildet werden (was auch praktisch nicht möglich wäre). .Sowohl Service Bus als auch Orchestrierungsengine können als Datenquellen für ein DWH in Frage kommen, wenn Daten über Servicenutzung (Monitoring) bzw. über das Laufzeitverhalten von Businessprozessen für die Analytik von Interesse sind.
Die Software zur Analyse von Daten, die auf DWH aufsetzen, wird primär lösungsspezifische Komponenten zum Datenzugriff und für die Businesslogik (hier: Analytik) einsetzen. Denkbar ist, es das bestimmte Analysen mir hoher geschäftlicher Bedeutung auch als Service verfügbar gemacht werden

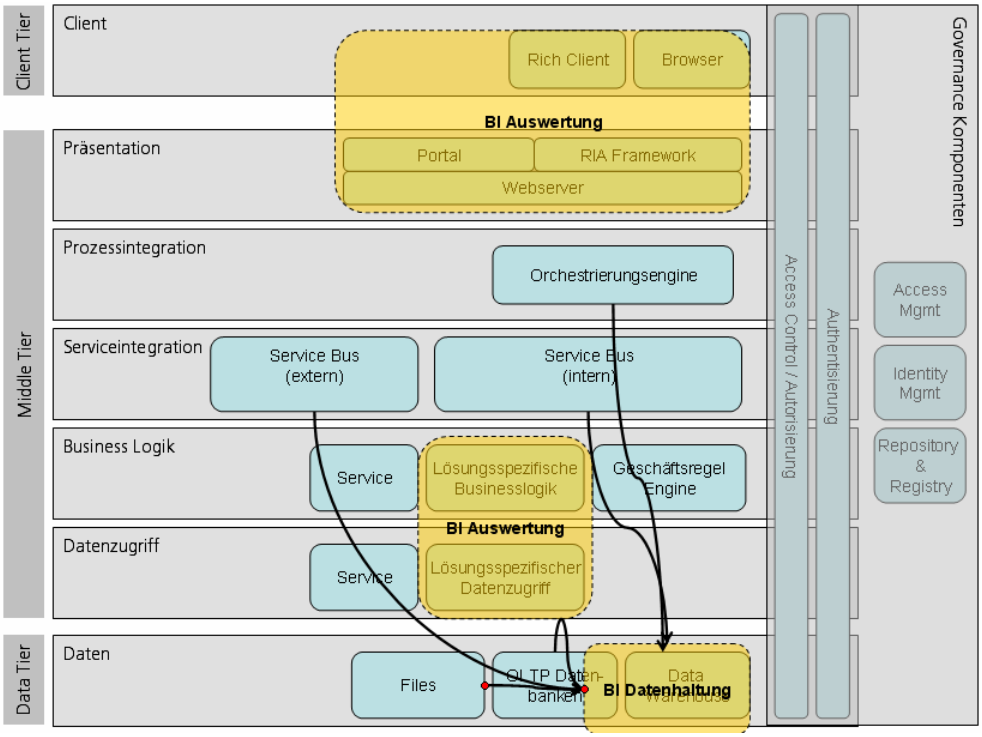

Abbildung 10: Abbildung von BI Lösungen in der Referenzarchitektur

3 Funktionale Patterns der technische Referenzarchitektur

3.1 Patterns für den Datenzugriff

3.1.1 Realisierung von „Adapter Services“ auf bestehenden Systemen

Nicht alle Daten sind über den Scope einer Lösung hinaus von Interesse., daher wird es immer (auch im Kontext einer weit entwickelten SOA) Komponenten zum lösungsspezifischen Datenzugriff geben. Wenn aber der Bedarf für eine Wiederverwendung von Datenzugriffslogik besteht, werden Services einfach durch die Hinzufügung eines öffentlichen Interfaces für eine Komponente realisiert.

Wichtig: Services dürfen auf lösungsspezifische Komponenten zum Datenzugriff im Kontext derselben IT-Lösung zugreifen. Der Zugriff auf lösungsspezifische („private“) Komponenten anderer Lösungen ist aber aufgrund mangelnder Transparenz abzulehnen. Siehe zu diesem Thema auch die analoge Diskussion im Abschnitt 3.2.

Für die Realisierung von Adapter Services gibt es zwei Grundansätze:

- Alternative 1: Native Implemenierung eines Adapter Service im Kontext des datenhaltenden Systems: Das datenhaltende System wird durch zusätzliche Schnittstellen so erweitert, dass auf die gewünschte Funktionalität über diese zugegriffen werden kann.
  Typisches Beispiel: Ein Webservice wird implementiert, der den Datenzugriff abstrahiert. Integration dieses WS dann über den Service Bus möglich.

- Alternative 2: Nutzung eines generischen Adapter-Frameworks (oft Teil eines Service Bus): Über einen generischen, konfigurierbaren Adapter wird direkt auf das datenhaltende System zugegriffen, wobei vorhandene Schnittstellen verwendet werden. Typisches Beispiel: Direktzugriff per Adapter auf eine Datenbank.

Um das SOA-Prinzip der Autonomie von Services (siehe [3]) realisieren zu können, ist das Lösungsmuster (1) meist vorzuziehen, da hier eine vollständige Kontrolle des Datenhaltenden Systems über die Servicefunktionalität erhalten bleibt.

Um das Lösungsmuster (2) zu implementieren, muss der Direktzugriff physikalisch möglich sein, und die entsprechenden Rechte müssen vorhanden sein. Dies ist in der Regel nur innerhalb eines internen Netzwerks möglich. Ebenso muss hier sehr darauf geachtet werden, dass bei schreibenden Zugriff die Datenintegrität in jedem Fall gewahrt wird.

Durch Policies zu regelnde Aspekte:

- Wie die Grenzen von IT-Lösungen (bzw. Applikationen) und damit die Bereiche festgelegt werden, innerhalb derer lösungsspezifische Komponenten von Services verwendet werden dürfen.

3.1.2 Datenbankzugriff durch Adapter Services

Die vorliegende SOA-Referenzarchitektur macht nur minimale Vorgaben bezüglich der Realisierung von lösungsspezifischem Datenzugriff; hier sind ggf. unabhängig von SOA "best practices" zu definieren.

Es ist allerdings eine sinnvolle Praxis, die Trennung von physikalischer und logischer Datenbankstruktur durch Implementierung eines Zugriffslayers anzustreben. Dieser kann sowohl applikationsseitig als auch datenbankseitig erfolgen.

- Applikationsseitig: durch Einsatz eines DB-to-Object Mapping Tools wie Hibernate oder Toplink
- Datenbankseitig: durch Einsatz von "stored procedures", views und anderen Möglichkeiten der Datenbank

3.2 Patterns für die Implementierung von Businesslogik als Services

Wie schon zuvor erwähnt, treten in einer SOA auf mehreren Ebenen auf, abhängig von deren Funktionalität. Ein Service kann allein Datenzugriff kapseln oder aber Businesslogik. Ein Service unterscheidet sich dabei primär durch ein öffentliches, wohldefiniertes Interface von lösungsspezifischen Komponenten, und durch die Einhaltung der in [3] definierten SOA Prinzipien.

Es ist also wichtig zu verstehen, dass die Einordnung in die Schichten der Referenzarchitektur nur nach technischen Kriterien erfolgt. Parallel dazu ist es sinnvoll, Services nach fachlichen Kritierien zu klassifizieren – wie es in Abschnitt 3.2.4 geschieht.

3.2.1 Private vs. öffentliche Komponenten und Services

Grundsätzlich gilt, dass nicht jede Funktionalität als Service exponiert werden kann und sollte [Glossar]. Services bedeuten immer einen gewissen technischen und auch fachlich / organisatorischen Overhead der nur dann gerechtfertigt ist, wenn ein Service auch tatsächlich über die Grenzen einer einzelnen IT-Lösung hinweg verwendet und wieder-verwendet werden kann[5].

Ein gutes modulares Design ermöglicht es, auf einfache Weise "private" Funktionalitäten zu einem späteren Zeitpunkt als Businesservice zu exponieren.
Dabei spielt es eine Roll, wo diese Funktionalität realisiert ist. Zur Realisierung von Services können lösungsspezifische Komponenten aus dem Kontext der jeweiligen IT-Lösung verwendet werden. Zusätzlich gilt das Prinzip, das eine Komponente jeweils nur andere Komponenten aus der gleichen oder einer niedrigeren Schicht verwenden darf.

Was genau unter eine „IT-Lösung“ zu verstehen ist, muss durch Policies exakt definiert werden. In der Regel ist der Begriff gleichbedeutend mit einer in einem Projekt individuell entwickelten und als abgegrenzte Entität betriebenen „Applikation“ [7].

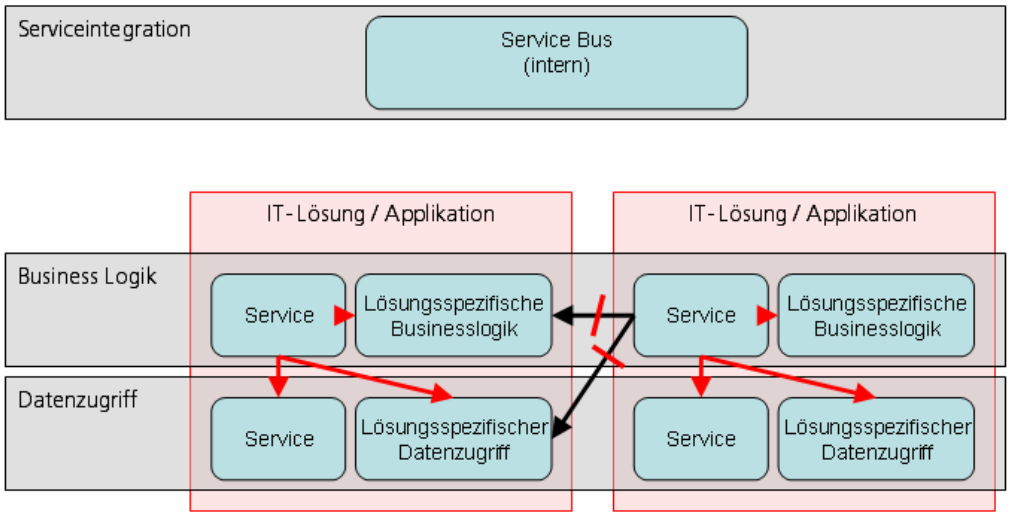

Abbildung 11: Nutzung lösungsspezifischer Komponenten nur innerhalb des Kontexts einer IT-Lösung

Im Gegensatz zur lokalen Rolle von Komponenten innerhalb von IT-Lösungen können Services über den Service Bus übergreifend konsumiert werden (siehe dazu auch 2.5.1). Services dürfen jederzeit auch von lösungsspezifischen Komponenten genutzt werden.
Es werden vor allem Business Services (im Sinne von [1] [5] [7]) über den Service Bus angeboten.

[5] Aus technischer Sicht sind typischerweise Funktionen mit sehr hohen Performanceanforderungen und/oder in Verbindung mir der Ein/Ausgabe oder dem Transport von sehr grossen Datenmengen nicht als Service prädestiniert.

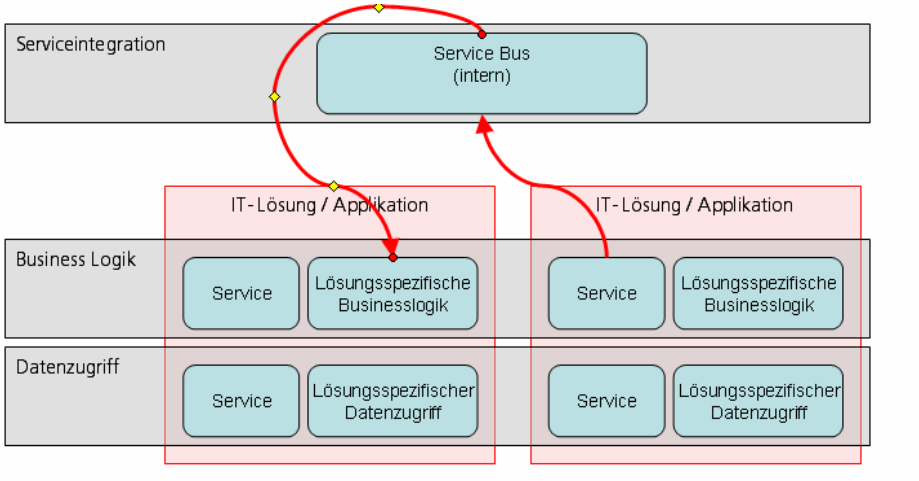

Abbildung 12: Prinzip der Nutzung von Services über den Service Bus (über Applikationsgrenzen hinweg)

Durch Policies zu regelnde Aspekte:

- Wie die Grenzen von IT-Lösungen (bzw. Applikationen) und damit die Bereiche festgelegt werden, innerhalb derer lösungsspezifische Komponenten von Services verwendet werden dürfen.

3.2.2 Architektur für die Realisierung von Services in IT-Lösungen

Wenn von einer Applikation ein Webservice zur Verfügung gestellt wird, liegt grundsätzlich die Auswahl der dabei genutzten Technologie in der Verantwortung des Owners des jeweiligen Systems. Allerdings müssen vorgegebene technische Standards und Richtlinien zur Realisierung unbedingt eingehalten werden.

Die grundlegenden Prinzipien bei der Implementierung von Service Providern und Service Consumern sind:

- die Businesslogik von der Realisierung des Messagings zu trennen; und
- die Realisierung von Security-Funktionen so weit als möglich dem Container zu überlassen.

Wenn die Realisierung eine Service Providers oder Consumers bereits bei im Rahmen der Entwicklung einer Applikation geleistet werden kann, so kann aus dem Messaging Interface direkt (über einen API call) auf die Businesslogik zugegriffen werden. Die Authentisierung von Usern (typischerweise gegen ein zentrales Identity Management) sollte dem Container überlassen werden. Auf jeden Fall eine Aufgabe des Containers sollte die Zuordnung der User Identität zu einem lokalen Rollen / Berechtigungsmodell sein. Falls feingranulare Autorisierung (siehe 3.9.5) notwendig ist, so muss aus der Businesslogik des Services auf die Rollen des Nutzers zugegriffen werden können. Mit anderen Worten: Der Rahmen für das Access Management wird vom Container für die Nutzung durch die Businesslogik des Services bereitgestellt.
Es muss aber auch möglich sein, bereits bekannte Identitäten, die mir der eingehenden Message transportiert werden, dem Container bekannt zu machen so dass dieser dann aktive Rollen / Berechtigungen ableiten kann. Für den Service muss es transparent sein, ob die Authentisierung lokal im Container erfolgt oder ob eine Identität aus einer eigehenden NAchricht verwendet wird, um Rollen / Berechtigungen zu bestimmen.

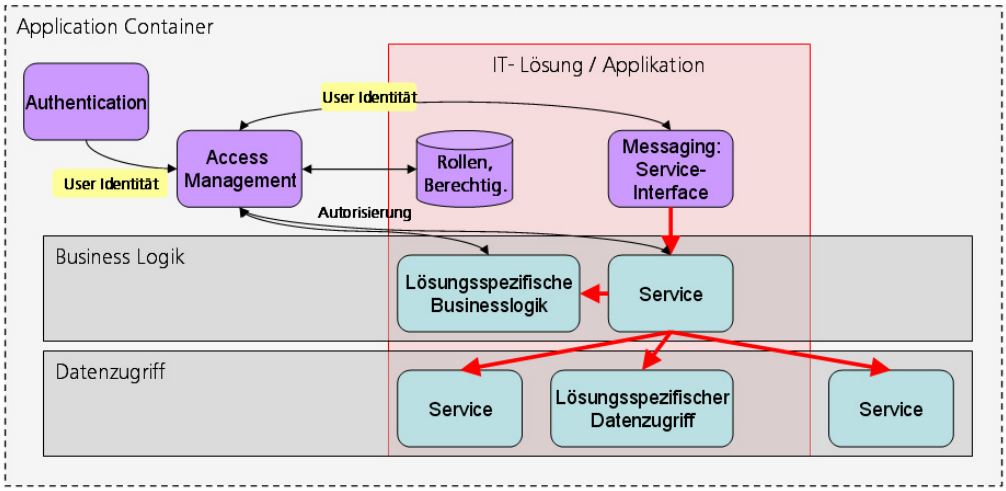

Abbildung 13: Architekturschema für die Realisierung von Services (hier: Service Provider)

Durch Policies zu regelnde Aspekte:

- Zu berücksichtigende technische Standards bei der Realisierung von Service Providern und Service Consumern

3.2.3 Realisierung von Services auf Basis bestehender Systeme

Es wird in der Verantwortung des Legacy-Systems selbst die Adapter-Funktionalität in Form eines Adapter-Services bereitgestellt. Dies kann dadurch geschehen, dass das System erweitert wird, oder aber das ein separates Modul bereitgestellt wird, das den Adapter-Service bereitstellt und auf proprietäre Art und Weise mit dem Altsystem kommuniziert. Man spricht auch von einem "Service-Wrapper" für das Altsystem. Auf dem Service-Bus sind dann nur noch logische Transformationen notwendig, um das Altsystem integrieren zu können. Wenn die Funktionalität des Altsystems ausreicht, kann auf diese Weise auch direkt ein Business-Service realisiert werden, der im Service Bus keine logische Transformation mehr benötigt.

Im Vergleich zur „nativen“ Realisierung eines Service Providers unterscheidet sich ein „Wrapper“ dadurch, das die Messaginglogik in einem anderen Applikations-Kontext oder sogar ein einem anderen Container bereitgestellt werden muss. Im letzteren Fall erfolgt die Anbindung an die Businesslogik über ein proprietätes Interface. Die Möglichkeiten zur Weiterleitung von Identitäten und damit zur feingranularen Autorisierung sind dabei limitiert und können ggf. nur durch proprietäre, Providerspezifische Logik realisiert werden.

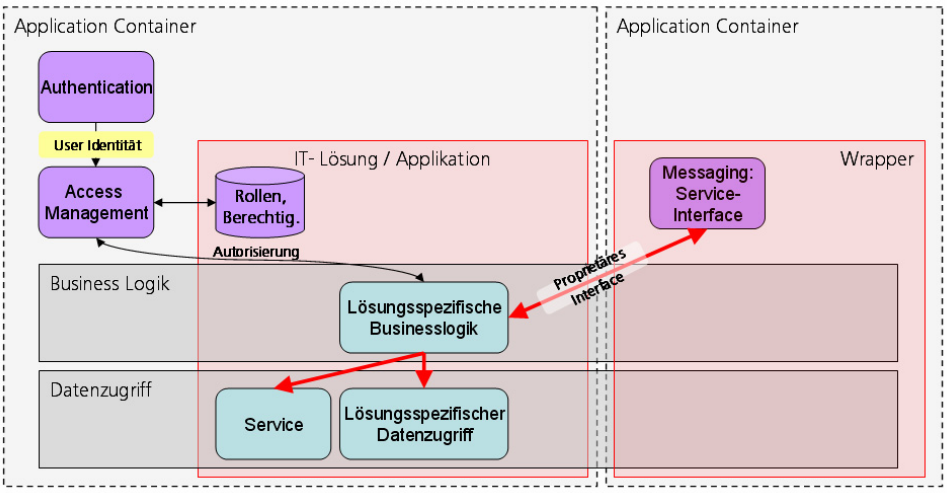

Abbildung 14: „Wrapper“ in einem eigenen Applikations-Container

Besser ist es, wenn möglich, den Wrapper und das Legacy-System im gleichen Container bereitzustellen. Dann ist ein Zugriff auf die Businesslogik durch API-calls möglich, und die Verwendung von weitergeleiteten Identitäten ist grundsätzlich nicht eingeschränkt.

Durch Policies zu regelnde Aspekte:

- Zu berücksichtigende technische Standards bei der Realisierung von Service Providern und Service Consumern

3.2.4 Scope von Services - Business Services

Im Zusammenhang mit der Realisierung von Businesslogik als Services ist eine entscheidende Frage, wo und von wem diese genutzt werden sollen. Übergreifend zu nutzende Services werden oft als „Business Services“ bezeichnet:

Business Services bilden "grobgranulare" geschäftliche Funktionalität ab. Was dies genau bedeutet, kann nicht auf technischer Ebene definiert werden, sondern hängt von der geschäftlichen Funktionalität ab. Die Vorgehensweise zur Identifikation geeigneter Business Services wird im Rahmen einer „Best Practice“ Empfehlung (siehe [5]) beschrieben.

In aller Regel kommen Services auf der Businesslogik-Schicht als Business-Service in Frage - umgekehrt ist aber nicht jeder Service auf dieser Schicht automatisch ein Business Service.

Im Kontext der Bundesverwaltung ist das Thema noch komplexes als in üblichen Szenarien in Unternehmen, da es mindestes drei verschiedene „Scopes“ für Services gibt [7]:

- Übergreifende eGovernment-Services, die in der ganzen Bundesverewaltung und ggf. sogar darüber hinaus genutzt werden können
- Services, die innerhalb eines Departements (bzw. innerhalb einer Businessdomäne, siehe dazu [1][5][7]) genutzt werden können
- Services mit nur lokalem Anwendungsbereich auf einer Ebene unterhalb des Departements – zum Beispiel in einer komplexen IT-Lösung

3.3 Patterns für die Implementierung von Businesslogik mit der Hife von Geschäftsregel Engines (BRE)

Die Verwendung von Geschäftsregel Engines (BRE) kann dann eine Alternative für die Realisierung von Services sein, wenn diese häufigen Änderungen unterworfen sind und ggf. von Fachexperten (und nicht von IT-Spezialisten) gepflegt werden müssen.

Es ist allerdings nicht im Scope dieser Referenzarchitektur, konkrete Entscheidungskriterien für den Einsatz von BRE zu machen.

Es gibt grundsätzlich zwei Alternativen für den Einsatz einer BRE:

- Variante 1: Spezifische Businessregel(n) als Service. Die auf der BRE bereitstehende Regel (bzw. ein Satz von Regeln) werden über einen Service konsumiert. In der Regel stellt die Business Rule Engine selbst die Funktionalität für die Bereitstellung eines messagorientierten Interfaces (d.h. meist Web Services) zur Verfügung.
- Variante 2: Spezifische Businessregel(n) als Komponente / als private Businesslogik. Die auf der BRE bereitstehende Regel (bzw. ein Satz von Regeln) werden als interne Komponente einer Applikation realisiert. Die BRE bzw. Komponenten derselben laufen dann im Kontext der Applikation.

Die Variante (1) ist dann sinnvoll, wenn die Businessregel(n) aus fachlicher Sicht einen guten Business Service darstellen; wenn also die Granularität geeignet ist und Wiederverwendung angestrebt wird. Diese Entscheidung kann nicht aus rein technischen Kritierien getroffen werden.

Aus technischer Sicht ist (1) sinnvoll, wenn der Service autonom agiernen kann. Das ist der Fall, wenn alle zur Abarbeitung der Regel(n) notwendigen Informationen im Serviceaufruf (Request Message) enthalten sind. Darüber hinaus sollte die Implementierung der Businessregel(n) wenn möglich immer frei von Seiteneffekten sein.
Wird hingegen zur Abarbeitung der Regel(n) ein Zugriff auf andere Datenbestände notwendig (die nicht wiederum selbst als Service gekapselt sind), so ist zu bedenken, dass aus einer BRE heraus in keinem Falle unkontrollierten (direkten!) Zugriffe auf Datenbestände in anderen Kontexten vorkommen sollten. Innerhalb des Kontexts einer IT-Lösung ist die direkte Nutzung von privaten Datenzugriffskomponenten durch die BRE aber kein Problem.

3.4 Patterns für die Serviceintegration (Service Bus)

Ein Service Bus bietet typischerweise folgende funktionale Patterns an:

1. Messageverabeitung
    a. Messaging: Versenden von Messages zwischen Serviceconsumern und Serviceprovidern. Unterstützung verschiedener Message-Exchange Patterns und Umsetzung zwischen unterschiedlichen Patterns (z.B. synchron / asynchron). Garantierte Zustellung von Messages.
    b. Routing: Zustellung von Messages an unterschiedliche Endpunkte (Service Provider) abhängig von Regeln. Beispiele: Dynamische Festlegung von Endpunkten (für Fail-over, load balancing), inhalts-abhängiges Routing
    c. Transformation: Logische Transformation verschiedener Messageformate in einander.
    d. Validierung: Prüfung des Formats bzw. des Inhalts von Messages gegen festegelegte Schemate (z.B. XML Schema); Reaktion auf allfällige Validierungsfehler kann festgelegt werden.
    e. Service Komposition: Zusammensetzung einer gewünschten Funktionalität aus mehreren feingranularen Einzelservices unter Einbeziehung von Transformationen. Es geht hier um kurzzeitige, in der Regel statuslose, voll automatisierte "Micro-Flows" im Gegensatz zu Businessprozessen / Workflows.
2. Konnektivität
    a. Protokollwandlung: Anbindung an unterschiedliche technische Protokolle. Konvertierung verschiedener phsikalischer Datenformate. Umsetzung von nicht service- und nicht messageorientierten Daten in Services. Beispiele: Webservice-, JMS-,File-, FTP-, Datenbankadapter
    b. Proxy für Services: Agieren als logischer Endpunkt sowohl für Serviceprovider als auch für Serviceconsumer. Der Service Bus kann so als logische zentraler Endpunkt auftreten, auf dem alle Business Services verfügbar sind - unabhängig vom Ort deren Implementierung.
3. Security Enforcement: Umsetzung von konfigurierten Sicherheits-Policies in Bezug auf die Authentisierung von Nutzer, die Autorisierung der Servicenutzung und der Servicebereitstellung, der Sicherheit auf Messageebene (Verschlüsselung, Signatur, etc.)
4. Konfigurative Festlegung der Eigenschaften des Sevice Bus:
    a. Statisch: Konfiguration der Eigenschaften in Punkto Konnektivität und Messaging, so dass ohne Veränderungen an Service Providern und Consumern die Logik des Routings modifiziert werden kann, z.B. also neue Consumer / Provider hinzukommen können.
    b. Dynamisch: Dynamische Veränderung der Konfiguration zur Laufzeit, so dass Veränderungen jederzeit und ohne erneutes Deployment vorgenommen werden können. Möglicher Bezug von laufzeitrelevanten Metadaten (z.B.: WSDL, Policies, ...) aus einem Registry oder einer anderen externen Quelle, anhand derer dynamisch die Konfiguration des Bus angepasst werden kann. So wäre es etwa im Extrem möglich, für jeden Serviceaufruf individuell das Verhalten des Service Bus anzupassen.
5. Management / Monitoring / Auditing:
    a. Überwachung der Funktionalität des Service Bus zur Laufzeit
    b. Möglichkeit des administrativen Eingreifens zur Laufzeit (z.B.: Routing-Regeln ändern, um einen Serverumzug zu ermöglichen)
    c. Protokollierung / Logging der Aktivitäten des Servic Bus in einstellbarem Detaillierungsgrad, optional auf sichere und verbindliche Art und Weise, um Audits in Bezug auf die Softwarenutzung zu unterstützen

Die wichtigsten dieser Patterns werden im Folgenden detaillierter dargestellt.

3.4.1 Einfaches Message-Exchange Pattern (MEP): Request / Reply (synchron)

Eine Anfrage eine Service Consumers (Request) wird unmittelbar vom Provider durch eine Antwort (Response) beantwortet. Tritt ein fachlicher Fehler auf, so kann auch eine Fehlernachricht (Fault) statt der Antwort verschickt werden. "Unmittelbar" in diesem Zusammenhang heisst einfach, dass in einer praktikabel kurzen Zeit und innerhalb einer „Session“ eine Antwort erfolgt. Was genau eine "Session" ausmacht, hängt einerseits von Benutzeranforderungen und andererseits von der technischen Realisierung auf einem Transportprotokoll ab. Bei einer Nutzung von HTTP ist die maximale Zeitdauer bis zu einer Antwort etwa durch den Timeout vorgegeben.

3.4.2 Einfaches Message-Exchange Pattern (MEP): Oneway, Fire-and-Forget (asynchron)

Eine Anfrage eines Service Consumers (Request) wird durch einen Provider entgegengenommen, ohne dass eine Antwort erfolgt. Dieses MEP ist die Grundlage für die Konstruktion aller komplexeren asynchronen Kommunikationspatterns, die aus mehreren Einzelnachrichten aufgebaut sind.
Ein Problem bei der Verwendung des Oneway-MEP ist das Fehlen jediglicher Bestätigung für den Erhalt der Nachricht durch den Provider. Es ist also eine Aufabe der Infrastruktur, eine "garantierte Auslieferung" sicherzustellen (siehe 3.4.8).

3.4.3 Komplexes Message-Exchange Pattern (MEP): Request / multiple Reply (asynchron)

Bei diesem MEP folgen auf eine Anfrage eines Service Consumers eine oder mehrere Antwortnachrichten, wobei eine grössere und vorher nicht fesgelegte Zeitspanne zwischen den Nachrichten vergehen kann. Die Kommunikation ist notwendigerweise asynchron.
Ein Beispiel für die Anwendung dieses MEP ist ein Service zur Auftragsabwicklung, der erst eine Bestellung entgegennimmt und dann immer wieder Statusupdates über den Fortgang des Auftrages zurückschickt. Die Realisierung dieses Patterns erfolgt durch Oneway-Nachrichten für jeden einzelnen Schritt. Dabei muss der Consumer, der den initialen Request versendet, für den Empfang der Folgenachrichten als Provider auftreten.
Dieses Vorgehen setzt voraus, dass das System, das den „initiale“ Consumer (später Provider) implementiert, über die gesamte Dauer des MEP verfügbar sein muss. Da dies oft schwer zu garantieren ist, muss der Service Bus alle Messages annehmen und ggf. zeitversetzt an die Provider versenden, wenn diese verfügbar sind. Das Problem der "garantierten Auslieferung" besteht hier ebenso wie beim Oneway-MEP (siehe 3.4.8).

3.4.4 Komplexes Message-Exchange Pattern (MEP): Publish / Subscribe

Beim Publish / Subscribe geht die Kommunikation immer vom Provider aus, wobei jeweils mehrere Consumer die Nachrichten parallel empfangen können. Das MEP enstpricht logisch der Rundfunksendung mit mehreren Programmen. Pro Programm (bzw. "Kanal" oder "Subject") werden Inhalte in unregelmässigen Abständen als Messages verschickt. Consumer können sich konfigurativ oder ggf. durch den Aufruf technischer Services für Programme anmelden und erhalten dann - bis zu einer Abmeldung -alle Nachrichten desselben.
Es ist wichtig, dass bei Publish / Subscribe die Kommunikation vom Provider und nicht vom Consumer ausgeht; in der Kosequenz muss also der Consumer stehts für den Empfang von Nachrichten verfügbar sein. Der Service Bus muss also in der Praxis auch hier die Aufgabe erfüllen, Messages zwischenzuspeichern und nur dann an Consumer auszuliefern, wenn diese auch verfügbar sind (siehe 3.4.8).

3.4.5 Statisches Message Routing

Für einen Service wird der physikalische Ort, an dem der zugehörige Service Provider erreichbar ist (der "Endpunkt") statisch konfiguriert. Ein Umzug des Servers erfordert also eine Konfigurationsänderung.
Bei der Verwendung von Webservices wird im WSDL Dokumender Endpunkt des Service Providers bei Verwendung des HTTP-Protokolls im Binding fest konfiguriert. Bei Verwendung eines Service Bus kann hier

immer ein fester Endpunkt (der Service Bus) eingestellt sein, wogegen der Endpunkt des eigentlichen Provider konfigurativ im Service Bus festgelegt wird. Dadurch tritt der Service Bus als Proxy auf und isoliert die Consumer von der Notwendigkeit, den physikalischen Server zu kennen, auf dem der Provider liegt.
Anmerkung: Der physikalische Ort des Service Consumers ist bei den meisten MEP dadurch gegeben, dass die Kommunikation durch diesen initiiert wird. Im Falle des Publish / Subscribe MEP sind allerdings die Endpunkte der Consumer wie Provider zu konfigurieren.

3.4.6 Content-Based Routing

Der Endpunkt für einen Service wird abhängig von Inhalten der Nachricht dynamisch festgelegt. Dies macht etwa dann Sinn, wenn mehrere Provider für einen Service zur Verfügung stehen, die unterschiedliche Varianten des Service implementieren. Aber auch im Rahmen komplexer Service-Komposition (siehe **Fehler! Verweisquelle konnte nicht gefunden werden.**) ist CBR wichtig.
Typischerweise wird auf der Basis von Informationen im SOAP-Header (nicht in der Payload) CBR betrieben.
Eine allerdings selten eingesetzte Variante ist das „Iterary Based Routing“, bei dem der Weg einer Nachricht durch eine Art „Laufzettel“ beschrieben wird. Nur wenige Produkte, wie etwa sedex und Sonic ESB, unterstützen dieses Pattern.

Ein Spezialfall von CBR ist die Unterstützung des Standards WS-Addressing durch den Service Bus. Dabei werden in einem standardisierten Header Informationen wie Empfänger, Absender, Returnadresse etc. festgelegt und so ein dynamisches Routing auf Standardbasis (und unabhängig vom zugrundeliegenden Transport) ermöglicht.
Anmerkung: Die Verwendung von WS-Addressing mit einem Service Bus ist zumindest bei der Verwendung von ReplyTo-Attributen problematisch, da dann ggf. Antwortnachrichten am Bus vorbei gehen könnten. Eine korrekte Verwendung von WS-Addressing in Kombination mit einem Service Bus erfordert daher in der Regel die Manipulation von Headern im Bus oder aber spezifische, propritäre Routinglogik.

3.4.7 Load Balaning und Failure Recovery durch dynamisches Routing

In diesem Spezialfall wird die dynamische Festlegung der Endpunkte benutzt, um entweder ein Load Balancing zwischen mehreren identischen Providern des selben Services zu realisieren, oder um bei Ausfall eines Providers automatisch auf einen anderen umzuleiten. Man unterscheidet oft zwischen XML- und DB-Konfiguration, je nachdem ob die Konfigurationsdaten in XML-Files oder in einer Datenbank abgelegt werden.
Anmerkung: Damit ohne Fehler von einem Provider zum anderen umgeschaltet werden kann, muss die Serivceimplementierung eine Reihe von Anforderungen erfüllen. Neben der in den SOA-Prinzipien [3] ohnehin geforderten Statuslosigkeit ist hier vor allem die „Idempotenz“ zu nennen: Mehrere nacheinander geschickte identische Request an einen Service dürfen nicht zu einem veränderten Resultat führen.

3.4.8 Garantierte Auslieferung von Messages

Der Service Bus garantiert, dass eine Nachricht mindestens einmal und / oder höchtens einmal ausgeliefert wird. Es wird - über einen geeigneten Kanal - eine Bestätigung der Auslieferung bzw. eine der Annahme der Message an den Sender zurückgegeben. Man kennt die folgenden Ausprägungen der garantierten Auslieferung von Nachrichten:

- AtMostOnce: Eine Nachricht wird höchstens einmal ausgeliefert
- AtLeastOnce: Eine Nachricht wird mindestens einmal ausgeliefert
- ExactlyOnce: Eine Nachricht wird genau einmals ausgeliefert

Dies kann – bei der Verwendung von Webservice – transparent für den Consumer erfolgen. Aus Sicht des Consumers muss dann aber ggf. die eigentlich den MEP „Oneway“ entsprechende Kommunikation als Request / Reply modelliert werden, um die Bestätigungsnachricht der Plattform verarbeiten zu können.
Weitere Alternativen für den Umgang mit der Bestätigungsnachricht:

- Alternativer Kanal für die Bestätigungsnachricht
- Anbindung des Consumers über ein anderes Protokoll als Webservices, welches Mechnismen für garantierte Auslieferung unterstützt

WS-ReliableMessaging

Der Standard WS-ReliableMessaging dient dazu, Funktionen für garantierte Auslieferung und Bewahrung der Reihenfolge von Nachrichten für Webservices zu ermöglichen, indem im SOAP-Header entsprechende Informationen kodiert werden. Dabei werden auf der Ebene der Nachrichten spezielle Signale wie ACK, NAK oder Anfragen für eine wiederholge Sendung definiert; ebenso werden die Nachrichten individuell numeriert. Mit WS-ReliableMessaging ist es daher Möglich, die in diesem Abschnitt beschriebenen Patterns auch auf der Basis von eigentlich unsicheren Protokollen wie HTTP zu realisieren. Die Umsetzung ist allerdings aufwendig und in fast allen Fällen weniger leistungsfähig als die Verwendung von sicheren Protokollen (z.B.: Queues). Für B2B-Anwendungen oder anderen Fällen, in denen ein Zugriff auf Queues nicht möglich ist, kann der Standard aber eine nützliche Alternative sein.

3.4.9 Entkopplung durch Zwischenspeicherung

Der Service Bus empfängt Nachrichten und speichert diese für eine gewisse Zeit, falls der Empfänger temporär nicht erreichbar sein sollte. Dadurch wird die notwendige Verfügbarkeit der Providersyssteme reduziert, und der Messageverlust wird vermieden.
Wichtig: Eine Bearbeitung der zwischengespeicherten Nachrichten kann natürlich nicht erfolgen; diese Lösung ist also nur bei asynchronen MEP sinnvoll.
In Zusammenhang mit der Zwischenspeicherung ist die Option zur garantierten Auslieferung („höchstens einmal“) besonders wichtig, da es leicht zu mehrfachen Auslieferungsversuchen kommen kann.

Der Standard WS-ReliableMessaging dient dazu, Funktionen für garantierte Auslieferung und Bewahrung der Reihenfolge von Nachrichten für Webservices zu ermöglichen, indem im SOAP-Header entsprechende Informationen kodiert werden.

3.4.10 Bewahrung der Reihenfolge von Services

Der Service Bus stellt sicher, dass Nachrichten in exakt der selben Reihenfolge ausgeliefert werden, in der sie abgeschickt worden sind. Dies erfordert, dass die Nachrichten entweder bereits auf Seite des sendenden Systems entsprechend codiert werden (z.B. mit fortlaufenden Nummern), oder aber das für die Einlieferung an den Service Bus ein Protokoll verwendet wird, das die Bewahrung der Reihenfolge sicherstellt[6].

3.4.11 Validierung von Nachrichten anhand von XML Schemas

In einer SOA müssen alls Nachrichten an und von Services einem wohldefinierten Interface entsprechen. Sowohl Consumer als auch Provider müssten also verifizieren, ob die generierten bzw. empfangenen Nachrichten tatsächlich der publizierten Interface-Definition entsprechen (= „Validierung“), da es sonst zu Laufzeitfehlern kommen kann. Eine Alternative dazu ist die Validierung der Nachrichten zentral durch den Service Bus.
In aller Regel wird XML Schema zur Definition er erlaubten Messageformate verwendet, so auch bei Webservices.

Der Service Bus validiert auf Wunsch (durch Konfiguration eingestellt) ein- und/oder ausgehende Nachrichten gegen hinterlegte XML Schema bzw. WSDL Dokumente. Die Verarbeitung wird abgebrochen und es wird ein Fehler erzeugt, wenn das Ergebnis der Validierung negativ ausfallen sollte.
Anmerkung: Die Validierung sehr grosser XML-Dokumente kann aufwendig sein und die Verarbeitung von Nachrichten spürbar verzögern. Das Problem kann, wenn notwendig, durch den Einsatz spezialisierter Hardware angegangen aber nicht grundsätzlich beseitigt werden. Es ist anzuraten, schon beim Service Design darauf zu achten, die Grösse von Nachrichten in einem vernünftigen Rahmen[7] zu halten und beim

[6] Webserver (z.B. Servlet Container wie Tomcat, alle üblichen Application Server) verwenden mehrere parallel laufende Threads, um eingehende HTTP Requests zu verarbeiten. Dies impliziert, dass nicht vorhersehbar ist, welcher Request von welchem Thread verarbeitet wird – es kann in Konsequenz dazu kommen, dass die ursprüngliche Reihenfolge der Nachrichten durcheinander gerät.
[7] Dieser ist durch die benutzte Soft- und Hardwareplattform bestimmt

Transport grosser Datenmengen die Patterns in Abschnitt **Fehler! Verweisquelle konnte nicht gefunden werden.** zu beachten.

3.4.12 Transport SOAP-over-HTTP(s)

Die ursprüngliche und als einzige voll standardisierte Form von Webservices verwendet als Transportprotokoll HTTP bzw. HTTPS. Die fachlichen Inhalte liegen dabei als XML-Dokument vor, dessen Struktur durch ein XML Schema beschrieben ist. Die Message wird in einem SOAP-Envelope verpackt, der einen "Header" mit technischen Informationen vom "Body"mit den fachlichen Inhalten trennt. Ein WSDL Dokument wird benutzt, um beide Aspekte zu beschreiben.
Der Service Bus muss die Verwendung von HTTP (und HTTPS) als Transport und den Umgang mit SOAP-Nachrichten beherrschen.

Anmerkung: Die Verwendung von HTTP als Transport hat neben den klaren Vorteilen, wie die nahezu universelle Akzeptanz und die Möglichkeit zur Überbrückung von Firewalls / Zonengrenzen auch Nachteile. So ist die garantierte Auslieferung von Nachrichten nur mit erheblichem Zusatzaufwand zu erreichen, da das Protokoll HTTP selbst in dieser Hinsicht keine Unterstützung bietet.

3.4.13 Protokollumsetzung

Der Service Bus muss Nachrichten über andere Protokolle als HTTP (und HTTPS), darunter vor allem JMS, MQ Series und andere etablierte Protokolle, empfangen und verschicken können. Der Service Bus muss auch Nachrichten von verschiedenen Protokollen aufeinander abbilden (mappen) können. Die Verwendung verschiedener Protokolle sollte für den jeweils nicht betroffenen Partner der Kommunikation transparent sein, soweit dies aufrund der Semantik des MEP möglich ist.
SOAP auf anderen Protokollen als HTTP(S): SOAP ist grundsätzlich unabhängig vom Transportprotokoll und kann daher auch zur Versendung von Messages über (praktisch beliebige) andere Protokolle verwendet werden. Zwar existieren wenige Standards in diesem Bereich, aber die Verwendung von SOAP[8] mit einem JMS-Interface zu einer Message-orientierten Middleware (MOM) oder auch die Kombination von SOAP direkt mit einem Queue Manager (wie Websphere MQ) ist gut etabliert. Für asynchrone MEP ist auch die Kombination von SOAP mit Email möglich (und sogar standardisiert), allerdings den Beschränkungen von Emails in Bezug auf die Leistungsfäigkeit unterworfen.
Bei der Verwendung von WS-* Standards, die Informationen im SOAP-Header abbilden (z.B. WS-Addressing) müssen diese ggf. zusätzlich auf proprietäre Header des zugrundeliegenden Protokolle abgebildet werden.

3.4.14 Verwendung proprietärer Protokolle

Unter proprietären Transportprotokollen verstehen wir alle, die nicht standardisiert bzw. weit verbreitet sind und für die daher keine Unterstützung durch den Service Bus von Haus aus erwartet werden kann. tSolche Transporte können zum Einsatz kommen, wenn sie bestimtme in einem fachlichen Szenario benötigte funktionale oder nichtfunktionale Eigenschaften bieten, die nicht ohne weiteres von anderen Lösungen erreicht werden.
Ein Beispiel im Bundesumfeld ist das System **sedex**, das einen sicheren und garantierten asynchronen Austausch von Dokumenten ermöglicht. Sedex kann logisch gesehen einfach als sicheres Transportprotokoll betrachtet werden.
Ein Service Bus muss so erweiterbar sein (durch Integration von eigener Logik), dass ein Abschluss an beliebige, proprietäre nachrichtenorientierte Protokolle möglich ist[9].

3.4.15 SOA Nachrichten mit Attachments

[8] SOAP bedeutet hier nichts anders als die Trennung von Nachrichten in einen Header und den eigentlichen Payload. Entgegen der urpsprünglichen Bedeutung des Akronyms hat SOAP nichts (mehr) mit der Serialisierung von Objekten zu tun.
[9] Im Falle von sedex, das Adapter für die Ein- und Auslieferung von einfachen Dateien anbietet, ist eine solche Eigenentwicklung allerdings nicht notwendig.

Die Datei wird -analog zum Vorgehen bei Emails - als Anhang (Attachment) an die SOAP-Message angehängt. Das Attachment wird dann einfach durch den Service Bus geschleust, ohne dass es in irgeneiner Weise dekodiert oder transformiert wird und werden könnte.
Es stehen zwei Standards zur Verfügung: Der auslaufende, aber noch verbreitete WS-Attachments und der kommende, aber noch nicht universell unterstütze MTOM. Auf der Ebene der Codierung der Nachrichten kommt dabei in beiden Fällen der Standard MIME (Multipurpose Internet Mail Extensions) zum Einsatz. Obwohl mit Attachments effizienter umgegangen werden kann als mit Inline-Codierung von binären Inhalten ist auch hier die maximale Grösse der zu verarbeitenden Nachrichten durch technische Schwierigkeiten begrenzt.

3.4.16 Alternativer Kanal für Filetransfer

Es wird für den Transfer grosser bzw. binärer Inhalte zwischen Consumer und Provider ein eigener technischer Kanal benutzt, der für diese Aufgabe optimiert ist. Dabei ist es möglich, dass die Signalisierung und der Austausch notwendiger Metadaten (z.B. Login-Daten) über den primären Kanal (z.B. Webservice) und über den Service Bus abgewickelt wird, und dass nur der reine Datentransport über den alternativen Kanal erfolgt.
Speziell für sichere, asynchronen und vorrangig dateiorientierten Datenaustausch bietet sich im Bundesumfeld die Nutzung von sedex als Kanal an.

Wichtig: Bei gutem Servicedesign sollte die Notwendigkeit zur Arbeit mit sehr grossen Messages so weit als möglich vermieden werden! Generell sind Services nicht dazu gedacht, direkt Massendatenverarbeitung abzubilden; speziell Webservices sind dazu technologisch auch nur sehr schlecht geeignet. Es muss daher immer auf einer fachlichen Ebene (bei der Business Service Modellierung) darauf geachtet werden, physikalischen Datentransfer und für die Service-Funktionalität unbedingt notwendige Daten voneinander zu trennen.

3.4.17 Generische Adapter für verschiedene physikalische Formate / Protokolle

Es wird, anstatt das Legacy System zu verändern, aus dem Service Bus heraus auf dieses zugegriffen, um Daten zu lesen und/oder zu schreiben. Beispiele dafür sind Datenbank-Adapter, File-Adapter oder auch Adapter für binäre APIs (in Java, .NET, COM, CORBA etc.), die sich mit dem Legacy-System verbinden und nach aussen hin über den Service Bus die Legacy-Funktionalität als Service anbieten können.
Einschränkungen von Datenbank-Adaptern: Funktionieren bei rein lesendem Zugriff generell gut, können aber beim Schreibzugriff erhebliche Probleme verursachen, da die Autonomie des Service ggf. nicht zu verwirklichen ist. Es muss sichergestellt werden können, dass ein Schreibzugriff nicht in Konflikt mit Aktivitäten des Altsystems tritt und dass keine undefinierten Zustände entstehen können. Dies ist nicht in allen Fällen möglich. Es ist auch notwendig, vollen Direktzugriff auf die Datenbank durch den Service Bus zu gestatten, was bei externen Systemen unmöglich und auch bei internen ggf. nicht unklritisch ist.
Einschränkungen von File-Adaptern:

- Da fileorientierte Schnittstellen meist durch regelmässiges Nachsehen ("Polling") überprüfen, um Daten zum Abholen bereitstehen, ergeben sich Verzögerungen, die für synchrone MEP meist nicht geeignet sind.

- Semantische Unterschiede der zugrundeliegenden Protokolle können nicht durch einen Protokolladapter kompensiert werden. Eine Umsetzung etwa von statuslosen auf statusbehaftete Protokolle; oder von Protokollen mit auf solche ohne Transaktionen ist prinzipiell nicht ohne Verlust von Funktionalität machbar.

Real verfügbare Service Busse / ESB bieten meist eine sehr grosse Zahl von solchen Adaptern für die verschiedensten Zwecke und Standards.

3.4.18 Transformation physikalischer Formate

Der Service Bus muss in der Lage sein, Dokumente in verschiedenen (non-XML) Formaten in XML umzuwandeln und umgekehrt solche Dokumente auch zu schreiben. Man spricht hier von physischer (im Gegensatz zu logischer) Transformation, da nicht die Inhalte sondern das Format derselben im Mittelpunkt

steht. Neben der reinen Funktionalität zur Transformation muss der Bus auch Möglichkeiten zum Design und zum Test der Transformationen bieten.

3.4.19 Logische Transformation

Sobald eine Nachricht einmal vom Service Bus physikalisch angenommen wurde, muss diese ggf. noch in ihrer logischen Struktur modifiziert werden. Der Service Bus ist in der Lage, XML-Dokumente durch Anwendung von XSLT-Transformationen oder anderen Mechanismen logisch zu transformieren. Neben der reinen Funktionalität zur Transformation muss der Bus auch Möglichkeiten zum Design und zum Test der Transformationen bieten.

3.4.20 Statuslose „Mikroflows“ zur Komposition von Services

Der Service Bus muss in der Lage sein, statuslose und kurz laufende Prozesse zu definieren, die koordinierte Ausführung der folgenden Patterns ermöglichen:

- Konsum von Services
- Formattransformationen, Anbindung an Legacy Systeme
- Protokollumsetzung (soweit in einem statuslosen „Microflow“ sinnvoll)
- Validierung Messages
- Transformation von Messages

„Koordinierte Ausführung“ bedeutet, dass Einzelschritte parallel und sequentiell durchgeführt werden können und dass Bedingungen ausgewertet werden können, die den Fluss innerhalb des Prozesses bestimmen.

„Statuslos“ bedeutet, dass keine Persistierung des Prozesses erfolgen kann, dass der Proess als Ganzes also „kurz laufend“sein muss – etwa innerhalb eines synchronen MEP auf der Providerseite.

Das Ergebnis der Service-Komposition ist ein Service, den der Service Bus als Provider anbieten kann. Die hierarchische Zusammensetzung mehrerer solche Service-Kompositionen muss möglich sein.

3.4.21 Grobgranulare Autorisierung durch den Service Bus

Der Service Bus muss eine grobgranulare Autorisierung auf der Ebene von Services bzw. von Serviceoperationen durchführen können. Dazu muss für den Nutzer, der einen Service nutzen will (mit definierter Identität) oder dieser einen bestimmten Service bzw. eine bestimmte Serviceoperation als Consumer aufrufen darf. Der Service Bus muss einen Service-Aufruf abbrechen und eine technische Fehlermeldung erzeugen, wenn die Autorisierung negativ ausfallen sollten.

Auf der Seite des Access Management ist also minimal eine Rolle pro definiertem Service notwendig, ggf. pro Operation dieses Services. Mittels einer Zuordnung von identifizierten Usern zu diesen Rollen kann entschieden werden, wer einen Service nutzen darf und wer nicht.

Anmerkung:Zusätzlich kann es sinnvoll sein, auch für die Berechtigung zum Angebot von Services bzw. Serviceoperationen als Provider eine analoge Autorisierung vorzunehmen. Dies kann aber nicht für reale Personen als Nutzer, sondern nur für technische Systeme geschehen. Ob es sinnvoll ist oder nicht, kann nur im Zusammenhang der realen technischen Plattform beurteilt werden.

3.4.22 Service Bus als Security Gateway

Der Service Bus muss in der Lage sein 1) eingehende Messages auf deren Konformität mit Sicherheitsrichtlinien hin zu überprüfen und 2) durch Konfiguration festzulegen, welche sicherheitsrelevanten Parameter auf welche Messages angewendet werden sollen.

1) Der Service Bus stellt fest, ob Inhalt und Header von eingehenden Messages mit der Konfiguration übereinstimmen und löst im Falle von Abweichungen einen technischen Fehler aus, bricht also die Kommunikation ab.

2) Es muss konfigurierbar sein, welche Anforderungen an Sicherheit für jeden Service bzw. einen „Ast“ der Kommunikation mit dem Service (also etwa nur die eingehende Nachricht vom Consumer) gestellt werden. Die Konfiguration sollte darüber hinaus auf den Bezug auf Rollen / Benutzergruppen erlauben, um den selektiven Einsatz von Sicherheitsfeatures zu ermöglichen.
3) Messages, die vom Service Bus ausgehen, werden automatisch konform zu den konfigurierten Sicherheitsanforderungen erzeugt. Dazu gehört insbesondere die Umsetzung der Codierung von Identitäten entsprechend der Vorgaben.

Beispiele:

- Es wird für einen Service festgelegt, dass interne User diesen mit Username / Passwort und „Basic Authentication“, externe User diesen aber nur mit einem verschlüsselten Serviceaufruf nutzen dürfen. Dies wird beim Eingang des Requests geprüft; die Response-Message wird automatisch entprechend angelegt.

- Eingehende Messages, in denen Identitäten mit SAML-Tokens codiert sind, werden verarbeitet. Die ausgehenden Messages werden mit Kerberos-Tokens versehen.

Diskussion: Dieses Pattern kann so spezifische Anforderungen an einen Service Bus stellen, dass spezialisierte Produkte als Soft- und/oder Hardware eine Alternative zur Nutzung des „normalen“ Service Bus sein können.

3.4.23 Monitoring, Logging, Auditing

Nach der Definition in diesem Dokument versteht man unter „Monitoring“ die Überwachung des Betriebszustandes der Plattform, und unter „Logging“ die laufende Dokumentation von Ereignissen im Betrieb zur späteren (zeitversetzten) Auswertung. Die folgenden Patterns sind also, je nachdem ob eine Live-Auswertung zur Betriebszeit möglich ist oder nicht, als Monitoring bzw. als Logging zu bezeichnen.

Funktionale Pattern: Monitoring / Logging

Die folgenden Varianten sind grundsätzlich wichtig. Nicht alle müssen von einer Plattform „out-of-the box“ unterstützt werden, ihre Implementierung muss aber möglich sein.

- Monitoring der Servicenutzung: Überwachung des Messages (nur Header) an und von Services.

- Payload Monitoring: Überwachung der Messages (Inhalte) an und von Services. Falls Verschlüsselung eingesetzt wird, ist die Auswertung nur den Besitzern der Schlüssel möglich.

- Performance Monitoring: Überwachung von Performancerelevanten Parametern wie Antwortzeit, Dauer der Verarbeitung in verschiedenen Komponenten, Durchsatz etc.

- Verfügbarkeits Monitoring: Überwachung der Verfügbarkeit von Serviceprovidern unabhängig von deren Nutzung. Sollte ein Provider nicht verfügbar sein, muss ein Alarm bzw. eine automatische Gegenmassnahme ausgelöst werden können.

- Pricing-relevantes Monitoring: Überwachung der Parameter, die für die Bepreisung der Nutzung von Services bedeutsam sind, darunter:

    - die Identifikation der Nutzer (Unique Ussers) von Services;

    - die Serviceaufrufe;

    - das transferierte Datenvolumen;

    - Anwendung eines Algorithmus auf Felder im Payload der Messages.

- Speicherung der Monitoring-Ergebnisse (ggf. Auditing-geeignet): Sichere, ggf. verschlüsselte und/oder signierte Speicherung der Ergebnisse. Im Falle von externen Audits muss die Speicherung so erfolgen, dass nachträgliche Änderungen nicht bzw. nur dokumentiert möglich sind.

- Komplexes / „custom“ Monitoring: Implementierung von komplexen, eigenen Verfahren zum Monitoring / Logging.

3.4.24 Service Bus Federation

Ein Einsatz verschiedener Service Bus Instanzen kann dann sinnvoll sein, wenn

- die Service Busse unterschiedliche technologische Stärken haben (Beispiel: unterschiedliche Busse für Java, .NET, SAP); oder wenn

- die Service Busse unterschiedliche nichtfunktionale Stärken haben (Beispiel: Ein dedizierter Service Bus als Security Gateway); oder wenn

- die Service Busse bestimmte funktionale Patterns mehr oder weniger gut realisieren (Beispiel: sedex als Service Bus für sichere und garantierte asynchrone Zustellung von Dokumenten).

Alternatives funktionales Pattern: Master-Slave Beziehung zwischen Service Bussen

Aus Sicht des Consumers ist transparent, über welchen Service Bus ein bestimmter Provider angeschlossen wird. Der Consumer sieht nur einen Bus, der den jeweiligen Service als Proxy bereitstellt. Dieser Service Bus leitet dann die Nachrichten an einen anderen Bus weiter, an dem der eigentliche Service Provider angebunden ist.
Diese Alternative ist vor allem dann sinnvoll, wenn für die Consumer-Seite keine unterschiedlichen Anforderungen bestehen und daher die Komplexität der verschiedenen Busse für den Consumer transparent bleiben soll. Würde etwa ein Service Bus B als Security Gateway für eine Kommunikation nach aussen verwendet, so ist dies für Consumer an Service Bus A nicht entscheidend und kann also „versteckt“ werden.

Alternatives funktionales Pattern: Parallelbetrieb von gleichberechtigten Service Bussen

Hier sind die Service Busse gleichberechtigt, und die Consumer wählen abhängig vom jeweils benötigten Service den entsprechenden Service Bus aus.
Da dies die Komplexität aus Consumersicht erhöht ist dieses Pattern nur dann sinnvoll, wenn die Anforderungen an die Anbindung an die Busse sehr unterschiedlich sind. Wenn zum Beispiel ein Service Bus B verwendet wird, der gerade für den fileorientierten Datenaustausch optimiert ist (wie etwa sedex), macht es wenig Sinn, das File zuvor erst in einen Webservice für den „Master-Service-Bus A“ zu verpacken.

3.4.25 Statische Konfiguration des Service Bus

Die folgenden Eigenschaften eines Service Bus müssen statisch konfiguriert werden können:

- Die Interfaces der angeschlossenen Services

- Die Endpunkte der angeschlossenen Service Provider

- Falls genutzt, die Funktionsweise der Adapter

- Falls genutzt, alle Arten von Transformationen, Validierungen und Servicekompositionen

- Routing-Regeln

- Die pro Serviceprovider und –Consumer zu nutzenden Sicherheitsmechanismen (Authentisierung, Datensicherheit)

- Die zugriffsberechtigten Rollen pro Serviceprovider (grobgranulare Autorisierung)
- Die gewünschte Art und Tiefe von Logging / Monitoring – Informationen

Eine Konfiguration ist dabei immer so lange gültig, bis diese durch eine neue Konfiguration ersetzt wird.

3.4.26 Dynamische Konfiguration des Service Bus

Über die statische Konfiguration hinaus ist es sinnvoll, bestimmte Eigenschaften des Service Bus auch dynamisch – also ggf. auch für jeden individuellen Nachrichtenaustausch – festlegen zu können.
Dazu muss der Service Bus Informationen über Service Endpunkte und/oder Security Policies von einer Service Registry beziehen können und diese sofort verwenden können.

3.5 Patterns für die Prozessintegration

Funktionale Lösungsmuster (Pattern):

Die formale Definition und Dokumentation von langlaufenden, statusbehafteten Businessprozessen in einer geeigneten Sprache ist notwendig. Der aktuelle Standard dafür ist WS-BPEL 2.0 (oft nur BPEL genannt). Nur zur Dokumentation bzw. zur grafischen Darstellung eignen sich BPMN und UML 2.x Activity Diagramme.

Über die Standards hinaus sind fast immer proprietäre Erweiterungen notwendig, da die Interaktionen mit Menschen (human tasks) dort nicht berücksichtigt sind.

Zur Prozessintegration in der Praxis ist eine Laufzeitumgebung ("Engine") notwendig, die den Ablauf der Prozesse steuert und Statusverwaltung / Persistierung übernimmt. Es kann sich, muss sich aber nicht, um eine physikalisch zentrale Einheit handeln.

Integration mit dem Service Bus: Im SOA Schichtenmodell wird davon ausgegangen, dass die Orchestrierung auf den Services aufsetzt. Das Konsumieren von Services aus dem Prozess heraus setzt also voraus, dass auf diese auf dem Service Bus zur Verfügung stehen. Minimal muss also eine Orchestrierungsengine Services (typischerweise Web Services) auf dem Service Bus konsumieren können, und ebenso Services für die Nutzung durch andere auf dem Bus bereitstellen können. Eine weitergehende Integration ist aber wünschenswert, um auch die Fähigkeiten des Service Bus im Bezug auf Konnektivität, Messageverarbeitung und Security voll nützen zu können[10]

Diskussion:
In der Praxis besteht oft die Notwendigkeit, dass die Interaktion mit Menschen als Teil von Businessprozessen einbezogen wird. Dies erfordert neben der Unterstützung langlaufender Prozesse auch den Umgang mit Benutzeroberfläche. Mangels verfügbarer Standards sind hier unterschiedliche, proprietäre Lösungen verfügbar. Es muss klar definiert werden, in welchen Fällen welche Technologien und Muster zur Erstellung von Benutzeroberflächen zum Einsatz kommen.

Das Thema "Orchestrierung" rückt typischerweise erst dann in den Fokus, wenn eine ausreichende Zahl von Business Services zur Verfügung steht. Der direkte Zugriff von der Orchestrierungsschicht auf Legacy-Businesslogik bzw. direkt auf die Datenhaltungschicht ist zwar mit vielen Produkten technisch möglich, hat aber grosse Nachteile aus Architektursicht und sollte vermieden werden.

Eine Vermischung der Gebiete des Business Process Management und der klassischen Workflows findet zunehmend statt. Es ist aber in der Regel schwerer, spezielle Workflow-Lösungen auf Basis einer eher generischen Orchestrierungsengine aufzubauen als mit einer spezialisierten Software. Die Integration von

[10] Viele reale Lösungen bieten stattdessen eine Duplizierung von vielen Funktionalitäten des Service Bus in der Orchestrierungsengine. Dies kann zu erheblichen Problemen führen und eine saubere SOA-konforme Architektur konterkarieren.

übergreifenden Businessprozessen mit typischen Workflows (etwa bei der Dokumentenbearbeitung) wird dadurch aber zu einem wichtigen Thema, das berücksichtigt werden muss.

3.6 Patterns für die Implementierung von Benutzeroberflächen in einer SOA

3.6.1 Webapplikationen und Rich Internet Applications

Bei der Implementierung von Benutzeroberflächen ist zwischen zwei grundlegenden Alternativen zu differenzieren:

1. Der Nutzung von Web-Applikationen, bei denen der Client über einen Browser auf die Anwendung zugreift der sowohl die Darstellung der Benutzeroberfläche als auch die Interaktion mit dem Nutzer übernimmr, und
2. Rich-Client Applikationen, bei denen die Darstellung der Benutzeroberfläche und die Interaktion mit dem Nutzer in einem speziellen, zur Applikation gehörenden Programm auf dem Client geleistet wird.

Webapplikationen können auf einen gut definierten und etablierten Satz von Standards aufsetzen, und auch die Sicherheitsfragen können gut adressiert werden. Die möglichen Oberflächen sind aber - was die Qualität der "User Experience" angeht - nicht mit Rich Clients oder RIAs zu vergleichen.

Als Mischform zwischen diesen beiden Extremen gibt es noch „Rich Internet Applications (RIA)“; bei denen eine in der Qualität der „User Experience“ mit Rich Clients vergleichbare Benutzeroberfläche in einem Browser ausgeführt wird.

Funktionale Lösungsmuster (Pattern) für Webapplikationen im Presentation Layer:

- Rendering von HTML-Seiten durch Mischen von statischen und dynamisch zur Laufzeit generierten Inhalten. Verschiedene Standards und Tool bieten verschieden mächtige Werkzeuge zur Gestaltung der Seiten (JSP, JSF, ASP.NET sind die wichtigsten Standards).
- Management von http/https Verbindungen, die stets von aussen initiiert werden (Standard in der Java-Welt sind hier Servlets) und Management des Kontexts einer Konversation mit einem User (Sessions).
- Erweiterte MVC-Architektur von Webapplikationen
    - MVC: Aufteilung einer Webapplikation in separate Komponenten für die Seitendarstellung (View), das zugrundeliegende Datenmodell (Model) und der Interaktionen mit der Seite (Controller). Verschiedene Frameworks (z.B. Struts, Spring, ...) stehen zur Verfügung, die den Aufbau eine Applikation nach diesem Prinzip erleichtern.
    - Der Zugriff auf Businesslogik wird innerhalb des "Models" abgewickelt bzw. von diesem abstrahiert. Dabei kann es sich sowohl um den Zugriff auf private Businesslogik als auch auf Business Services handelt.
    - Erweitertes MVC: Zusätzliche Schicht, die den Konsum und ggf. auch die Bereitstellung von Services abwickelt.
- Produktivität: Codegenerierung und Wiederverwendung von GUI-Elementen[11]

Funktionale Lösungsmuster (Pattern) für Rich Internet Applications im Presentation Layer:

Es gibt aktive Softwarekomponenten, die auf dem Applikationsserver (Backend) ablaufen und aktive Softwarekomponenten, die im Browser ablaufen. Hier die Backend-Seite:

[11] Generell, und unabhängig von der konkret gewählten Technologie bzw. dem gewählten Framework, ist der Aufwand für die Implementierung von Webapplikationen sehr hoch. Ein grosser Teil des Aufwandes kann aber potentiell durch die automatische Generierung von Code und anderen technischen Artefakten eingespart werden.

- Initiale Distribution der Seiten (inkl. des Codes für die Client-seitige Logik)
- Rendering der Oberfläche Client-seitig und/oder auf dem Backend
- Handling und Validierung von Nutzereingaben
- Asynchrone Kommunikation mit dem Backend
- Typischerweise proprietäres Messaging zwischen Browser und Backend
- Verarbeitung von Events, die Businesslogik aufrufen
- Aufruf der Komponenten, die die Businesslogik implementieren (Services oder private Komponenten)
- Asynchrone Kommunikation mit dem Client

Diskussion:
RIA hat zur Folge, dass auf dem Client über die reine Browserfunktionalität hinaus Applikationslogik ausgeführt wird. Das ist in jedem Falle bedeutsam aus der Security-Perspektive. Die verschiedenen auf dem Markt verfügbaren Lösungen sind in dieser Hinsicht teilweise unproblematisch; teilweise aber auch sehr kritisch. Neben der Produktauswahl sind Policies vorzusehen, die den Einsatz im Rahmen der Sicherheitsvorgaben festlegen.
Der Einsatz von RIA Frameworks darf nicht einfach mit Technologien für „schöne“ GUI gleichgesetzt werden. Auch die Abwicklung der (asynchronen) Kommunikation mit Services und anderen Komponenten mit Businesslogik spielen eine grosse Rolle und können für den Einsatz einer solcher Komponente sprechen.

3.6.2 Portale

Unter einem Portal versteht man eine Komponente, die die kontrollierte Nutzung mehrer Webapplikationen unter einem logischen „Dach“ ermöglicht.
Es steht eine grosse Auswahl von Portalprodukten zur Verfügung, die die verfügbaren nur rudimentäten Standards alle durch proprietäre Implementierung von Portalfunktionalität ergänzen.
Im Bereich RIA gibt es Ansätze für portalähnliche Funktionalitäten; umgekehrt können bestimmte Technologien aus dem RIA-Bereich auch von manchen Portalen unterstützt werden. Eine Kombination von RIA- und Portaltechnologien ist eine besondere Herausforderung und erfordert immer produktspezifische Lösungen und Vorgaben.

Funktionale Lösungsmuster (Pattern) für Portale :

- Einheitliche Zugangskontrolle (User Authentisierung, Autorisierung)
- Einheitliches Modell für die Navigation zur Auswahl aus den zur Verfügung stehenden Webapplikationen
- Rendering des Outputs der Webapplikationen, so dass diese im Portal genutzt werden können
- Mechanismus für den Datenaustausch zwischen verschiedenen Webapplikationen (manuell, teilautomatisiert, automatisiert)

Durch Policies zu regelnde Aspekte:

- Auswahl der über Standards hinausgehenden Features eines Portals
- Einsatzszenarien für Portale, Abgrenzung zur innerhalb von Webapplikationen zu entwickelnde Funktionalitäten

3.7 Patterns für Clients

Generell bieten Rich Clients gegenüber Webclients drei entscheidende Vorteile:

- Die Qualität der Benutzeroberfläche, und
- die Möglichkeit der Integration mit anderen lokalen Applikationen und Ressourcen, und

- die Möglichkeit des "Offline" Betriebs ohne Verbindung mit dem Backend

Dem stehen die Nachteile gegenüber, dass Rich Clients in einem aufwendigen Verfahren ausgerollt (also auf die Clients verteilt) werden müssen, und das dies bei jeder neuen Version wiederholt werden muss. Dies kann bei einer grösseren Zahl von Applikationen und Clients ein prohibitiver Nachteil sein, der zu ausserordentlich hohen Betriebs- und Wartungskosten im Vergleich zu Webclients führt. Der Einsatz von RIA-Lösungen kann ein möglicher Kompromiss sein, wenn man auf aufwendiges Ausrollen verzichten aber doch die drei oben genannten Nachteile weitgehend minimieren will.
Aufgrund diverser subtiler Unterschiede zwischen verschiedenen Browser-Implementierungen muss, um ein kontrolliertes Environment zu gewährleisten, eine Einschränkung sowohl der benutzten Produkte als auch der erlaubten Plugins/Erweiterungen durchgesetzt werden. Dies ist für den Fall der externen Nutzung von Funktionen über das Internet (eGoverment) allerdings nur sehr eingeschränkt möglich.

Besondere Aufmerksamkeit muss der Möglichkeit der Ausführung von Code auf dem Browser (JavaScript, ActiveX) gewidmet werden, die einerseits erweiterte Möglichkeiten für Webapplikationen bietet, andererseits aber im Hinblick auf Sicherheitsfragen reguliert werden muss.

Funktionale Lösungsmuster (Pattern) für Rich Clients:

keine Vorgaben im Rahmen dieser Referenzarchitektur

Rich Clients sollten aufgrund der oben beschriebenen Nachteile nur in wohlbegründeten Ausnahmefällen eingesetzt werden. Für eGovernment Anwendungen verbieten sie sich in aller Regel.

Funktionale Lösungsmuster (Pattern) für Clients von Webapplikationen und RIA:

- Aktive Komponenten im Browser: Businesslogik (typischerweise Verarbeitungslogik), die im Browser abläuft und in gewissem Mass die lokale Abarbeitung von Funktionalität ermöglicht. Einsatzgebiete sind etwa die Validierung von Nutzereingaben, das Handling von Nutzerinteraktionen, das Management asynchroner Verbindungen zum Backend und (eingeschränkt) die Interaktion mit lokalen Ressourcen[12]
- Zusätzliche Funktionalitäten über Plugins: Standardmässig nicht unterstützte Technologien, etwa für den RIA-Bereich, können über Browser-Plugins nachträglich installiert werden. Es ensteht damit aber das Problem der Softwareverteilung und der Versionierung der Plugins.
- Integration mit lokalen Applikationen über "Bordmittel" des Betriebssystems: Datenaustausch über "Copy & Paste" oder über das Filesystem; dies liegt völlig in der Verantwortung des lokalen Nutzers und muss vom Backend-System nicht explizit unterstützt werden.
- Direkte / automatisierte Integration mit lokalen Applikationen: Eine direkte Integration mit lokalen Applikationen ist im vollen Umfang nur mit einem Fat-Client Ansatz möglich; mit Einschränkungen und bestimmten Technologien (z.B.: ActiveX, Java Applets) auch aus im Browser laufenden Anwendungen heraus.

3.8 Übergreifende Patterns: Deployment in die Netzwerkinfrastruktur des Bundes

3.8.1 Deployment in den verschiedenen Zonen

Bei der Realisierung von SOA-Plattformen im Umfeld der Bundesverwaltung ist ganz wesentlich zu berücksichtigen, dass ein Ziel der SOA im Bundesumfeld (eGovernment!) die Integration von Systemfunktionalitäten über die Verwaltung hinaus ist. Es geht dabei sowohl um "interne" (also in der Bundesverwaltung betriebene) und "externe" (sogar nicht in der Bundesverwaltung betriebene) Systeme. Für ein genaues Bild ist das Zonenkonzept des BIT zu berücksichtigen [11], das verschiedene, voneinander abgegrenzte Netze definiert. Die Übergänge zwischen diesen Zonen finden in einer spezialisierten „internal transition zone“ (ITZ) statt. Zugriffe auf bzw. über das Internet werden über eine eigene demilitarisierte Zone

[12] Theoretisch kann auch sehr komplexe Businesslogik auf dem Browser ablaufen, so dass die Grenze zwischen Web- und Fat-Client verwischt. Der Zugriff auf lokale Ressourcen ist jedoch aus dem Browser heraus eingeschränkt, und auch Performanceaspekte verhindern in der Praxis allzu komplexe "Browser-Applikationen".

(DMZ / ETZ) geleitet. Das Büronetzwerk der Bundesverwaltung (KOMBV) ist (wie auch andere „private“ Netze von Departements) gegenüber Zugriffen von aussen stark abgeschottet; also aufgrund der Vorgaben zur Netzwerksicherheit [10] grundsätzlich nicht dazu geeignet, Systeme zu betreiben, die intensiv mit der Aussenwelt kommunizieren (z.B. eGovernment Anwendungen).
.Anwendungen, die per Design mit mehreren Zonen kommunizieren müssen, sollten daher in eine spezialisierte „Shared Service Zone“ (SSZ) plaziert werden, deren Sicherheitskonzept für moderne interaktive Anwendungen optimiert ist.

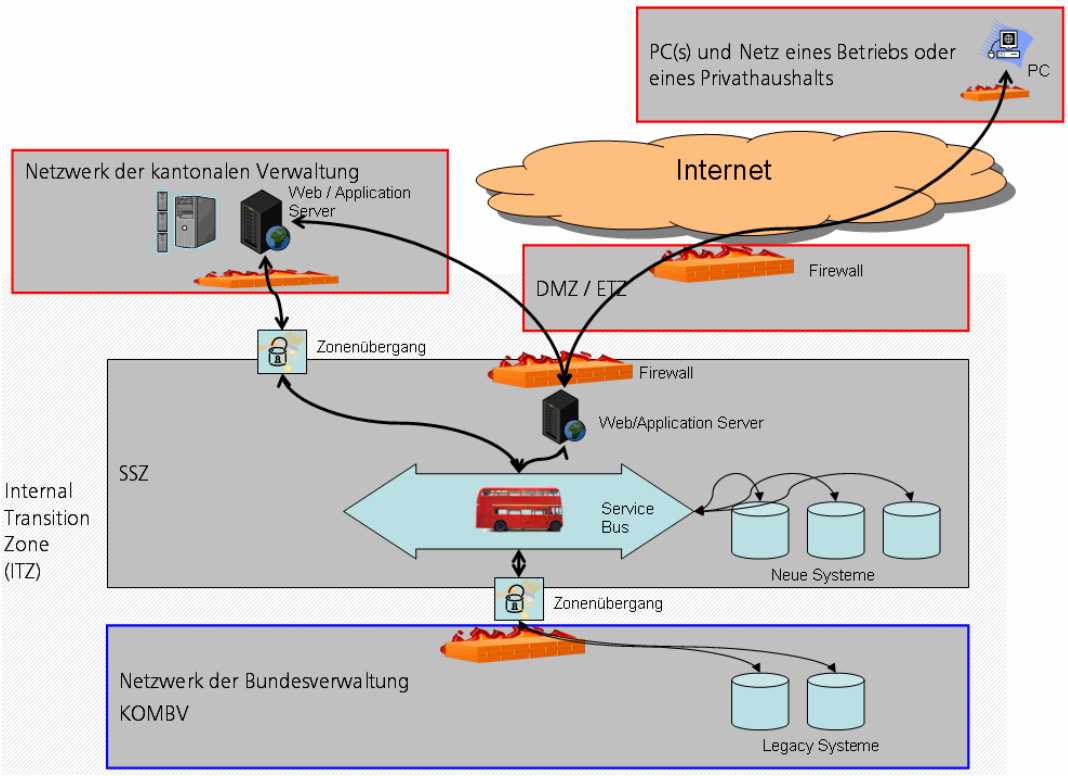

Abbildung 15: Skizze zu den verschiedenen Netzwerk-Zonen und den Übergängen zwischen diesen

Für die Refernenzarchitektur des EVD bedeutsame Systeme können also in verschiedenen Netzwerkzonen liegen. Man muss unterscheiden:

- Systeme in einer SSZ der Bundesverwaltung[13]
- Systeme im Büronetzwerk der Bundesverwaltung (KOMBV)
- Systeme in getrennten Zonen von Departements der Bundesverwaltung (z.B. EJPD)
- Externe Systeme, die direkt inter „internal transition zone“ der Netze der Bundesverwaltung angebunden werden (z.B. kantonale Systeme)
- Externe Systeme (ausserhalb der Bundesverwaltung), die über das öffentliche Internet angebunden sind

Die meisten neuen Systeme, vor allem im eGovernment Umfeld, werden also in der SSZ betrieben werden müssen. Daneben werden aber – zumindest in einer Übergangszeit – von der SSZ aus auch Zugriffe auf Legacy-Systeme im KOMBV erfolgen müssen, da eine grosse Zahl von Systemen derzeit dort betrieben wird und nicht kurzfristig neu deployed werden könnnen.

3.8.2 Zonenübergänge

Es wird nun - im Sinne der Vorgaben zur Netzwerksicherheit des BIT [5] -angenommen, dass jedigliche Kommunikation über Zonengrenzen hinweg in aller Regel über die Protokolle http und https erfolgt.

[13] Eine Netzwerkzone, die die Anforderungen zur Netzwerksicherheit in [11] erfüllt. Derzeit existiert eine SSZ des BIT, weitere SSZ-Instanzen sind in Planung.

Dabei sind nur die folgenden Varianten zu unterscheiden, da jeweils die Sicherheitsregeln der Zone in der die Kommunikation initiiert wird, zum Einsatz kommen

- Zonenübergang in in die SSZ
  - Zugriff auf Webapplikationen (http oder https), die von Systemen in der SSZ angeboten werden. Dabei ist ggf. noch zwischen anonymer und nichanonymer Nutzung und nach Zugriffen aus dem Internet und aus anderen Zonen zu unterscheiden.
- Konsum von Webservices, die von Systemen in der SSZ angeboten werden
  - Sicherer Zonenübergang aus dem KOMBV in die SSZ
-
- Zugriff auf Webservices, die von Systemen im KOMBV (als Service-Provider) angeboten werden
  - Sicherer Zonenübergang von der SSZ zum Netz der Bundesverwaltung (KOMBV)
- Zugriff auf von kantonalen Systemen angebotene Webservices aus der Bundesverwaltung heraus
  - Zonenübergang von der SSZ in externe Zonen über das Internet
- Zugriff auf von externen Systemen angebotene Webservices aus der Bundesverwaltung heraus
  - Zonenübergang von der SSZ auf kantonale Systeme

3.9 Übergreifende Patterns: Zugriffskontrolle

In diesem Abschnitt werden Authentisierung, Autorisierung, Access Management und Identity Management behandelt.

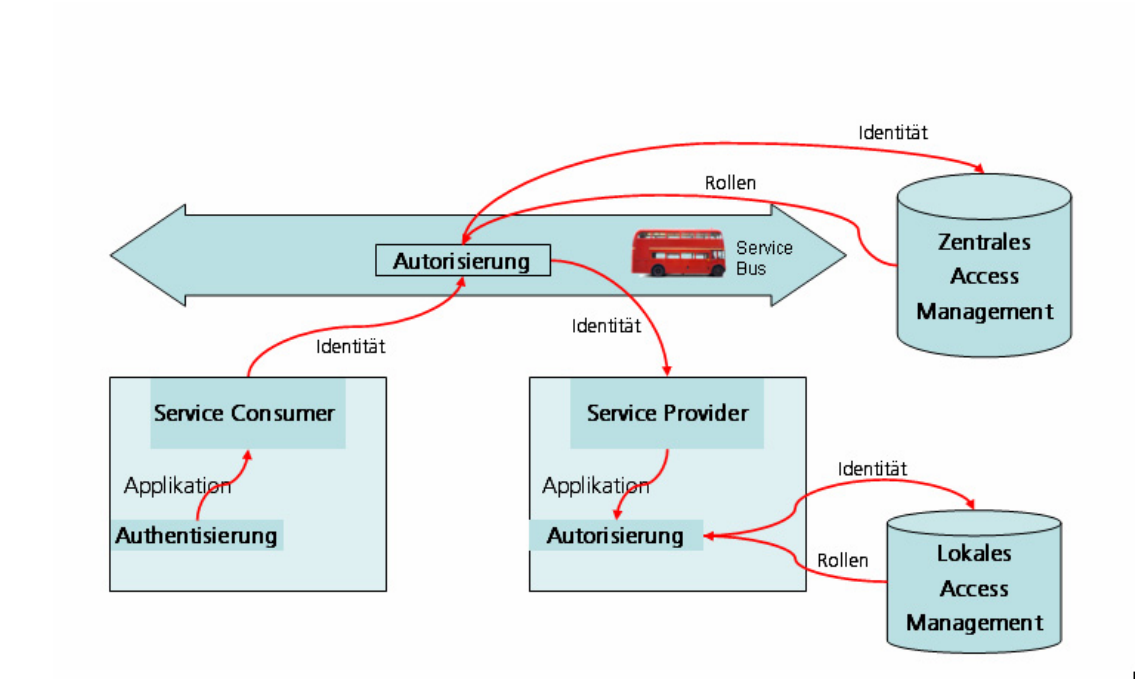

Abbildung 16: Übersicht über Weiterleitung von Identitäten, zentrale grobgranulare und lokale feingranulare Autorisierung

3.9.1 Lokale Authentisierung gegen ein zentrales IDM

Die Authentisierung findet jeweils lokal dort statt, wo der Zugang zu einer Komponente beschränkt werden muss. Dazu werden lokal User Credentials (etwa Username / Passwort; ggf. biometrische Merkmale) bezogen und gegen das zentrale Identity Management (IDM) geprüft. Dadurch wird die Identität des Nutzers eindeutig festgestellt. Dabei muss sichergestellt werden, dass die Anfrage beim IDM über einen sicheren Kanal und ausschliesslich durch bekannte interne Systeme erfolgt
Eine Prüfung gegen die zentrale Stelle kann nur in solchen Fällen entfallen, wo eine Komponente ausschliesslich in lokalem Kontext genutzt wird; also etwa nie in Zusammenhang mit SOA relevant wäre.

Je nach den Anforderungen an Sicherheit wird stets eine lokale Authentisierung von individuellen Personen gefordert, oder aber es können auch Nachrichten von bestimmten Systemen pauschal authentisiert werden.

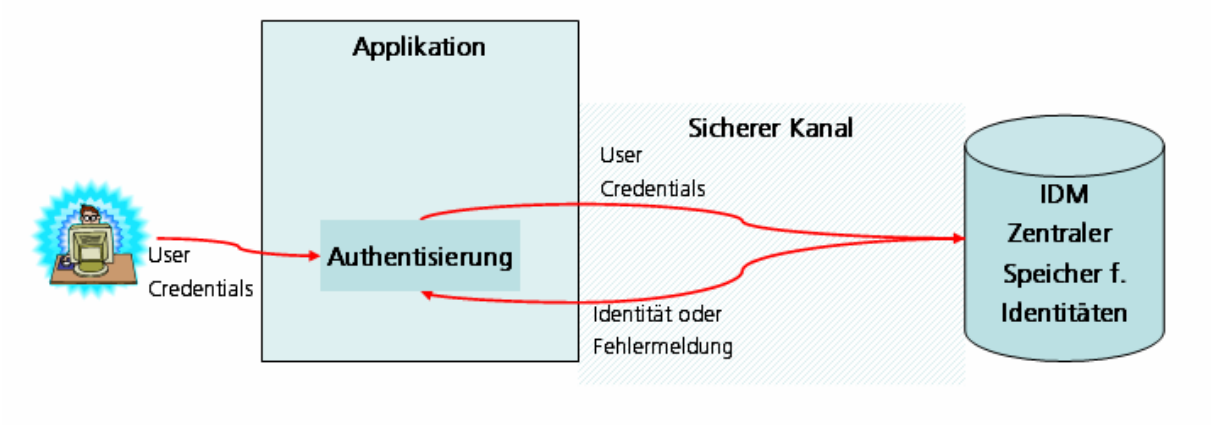

Abbildung 17: Schema der lokalen Authentisierung gegen ein zentrales IDM

3.9.2 Zentrales IDM

Das Identitiy Management hat für das ISCeco mehrere Aspekte, die ggf. mit unterschiedlichen technischen Ansätzen adressiert werden müssen:

- Verwaltung der Identitäten der internen Nutzer im Bund
- Verwaltung der Identitäten von Systemen in diversen internen und externen Netzen, mit denen eine Kommunikation erfolgt
- künftig auch von Bedeutung: Verwaltung der Identitäten externen Nutzer von Applikationen (z.B.: Nutzer eines Portals)

In allen Fällen müssen die Identitäten den Systemen, die User authentisieren, über einen sicheren Kanal zur Verfügung stehen.
Parallel müssen geeignete User Interfaces zur Pfelge der Identitäten bereitgestellt werden, die alle Aufgaben über den Lebenszyklus individueller Nutzer (Anlegen, Mutieren, Entfernen) unterstützen müssen. Die konkrenten Anforderungen funktionaler und nichtfunktionaler Art an solche Lösungen zur Verwaltung und Pflege von Identitäten sind allerdings nicht Gegenstand dieser Referenzarchitektur. Entsprechende Lösungen existieren bei der Bundesverwaltung bereits bzw. werden neu entwickelt.

Eine zu berücksichtende Alternative bzw. Erweiterung ist in 3.9.7 beschrieben.

3.9.3 Weiterleitung von Identitäten

Nach der Authentisierung (der Identifikation eines Nutzers) muss es möglich sein, die nun festgestellt Identität auf sichere Art und Weise anderen Komponenten der SOA Plattform zur Verfügung zu stellen. Dies ist unerlässlich, um unnötige mehrfache Authentisierungen zu vermeiden und ein effizientes Zusammenspiel der gesamten Plattform zu ermöglichen. Die Identität muss also als Teil einer Nachricht versendet werden können ohne deren eigentlichen Inhalte („Payload“) zu beeinträchtigen. Bei Webservices sollten die Identitäten also im SOAP-Header codiert werden.
Die Informationen über Identitäten müssen zwischen Service Bus und allen beteilgten Service Consumern und –Providern weitergeleitet werden können. Sollte für ein „Leg“ der Kommunikation keine Identität notwendig sein, so muss die Weiterleitung für dieses selektv per Konfiguration abgestellt werden können.
Bei der Weiterleitung von Identitäten muss garantiert werden können, dass a) nur die vom zentralen IDM gelieferten Identitäten verwendet werden, und dass b) Identitäten nicht unautorisiert modifiziert werden dürfen. Dazu kann es notwendig sein, die Identitäten (etwa im SOAP Header) zu signieren oder sogar zu verschlüsseln; man spricht dann von sog. „Tokens.
Ein verbreiteter Standard dazu ist SAML. Mit Hilfe von SAML Tokens können Identitäten manipulationssicher codiert werden; neben den Identitäten können zusätzlich sogar Rollen- und Rechteinformationen

transportiert werden. Alternativ können – zumindest in der Windows-Welt – sog. Kerberos-Tokens verwendet werden.

3.9.4 Grobgranulare Authentisierung im Service Bus

siehe 3.4.21

3.9.5 Feingranulare Autorisierung durch die Applikations-Container

Über die bisher beschriebene **grobgranulare Autorisierung** hinaus wird in der Regel auch eine sogenannte **feingranulare Autorisierung** notwendig sein. Dabei handelt es sich um die Autorisierung einer Aktion auf der Basis der Inhalte eine Nachricht (man spricht manchmal auch von „row-level autorization“)
Beispiel: Ein Service "Lesen Kontostand" kann von allen Mitarbeitern eine Call-Centers genutzt werden, aber nicht für eine bestimmte Gruppe von Konten mit höhen Sicherheitsanforderungen. Die Entscheidung zur Gewährung des Zugriffs hängt also von der Kontonummer ab.

Eine feingranulare Autorisierung in diesem Sinne kann nicht konfigurativ erfolgen, sondern nur aus dem Code heraus. Dazu muss die Identiät der Implementierung des Services bzw. dem Container, in dem die Implementierung läuft, zur Verfügung stehen. Im Container kann die Identität dann (lokalen) Rollen zugeordnet werden. Die Autorisierung kann dann durch Abfragen auf die Zugehörigkeit zu Rollen innerhalb der Serviceimplementierung gemacht werden. Beispielsweise bieten J2EE Container (Application Server) standardisierte Möglichkeiten zur Definition von Rollen und zur Abfrage derselben aus Applikationen heraus. Es muss einzig sichergestellt sein, dass die Nutzeridentität dem Container bekannt ist, um diese Features nutzen zu können.

Diskussion: Bis zu einem gewissen Grad kann die Notwendigkeit zur feingranularen Autorisierung durch den Schnitt der Services vermieden werden. In obigem Beispiel: Mit zwei Services „lesenKontostandNormal“ und „lesenKontostandVIP“ würde allein mit der grobgranularen Autorisierung ausreichen. Dies ist aber sicherlich nicht in allen Fällen möglich (speziell nicht, wenn komplexere Logik zur Autorisierung notwendig ist) und es führt auch zu Mehraufwand bei der Verwaltung der Services.

3.9.6 Zentrales Access Management

Grundsätzlich werden durch SOA keine speziellen Anforderungen an Mechanismen zur Autorisierung bzw. an Rollen- und Rechtemodelle festgelegt, die über Bekanntes hinausgehen. Im Gegenteil, die (grobgranulare) Autorisierung von Services setzt nur ein recht einfaches Rechtemodell voraus. Notwendig ist:

- Die Festlegung von Rechten, die zur Nutzen einer Serviceoperation bzw. eines Services als Consumer autorisieren
- Optional sind auch Rechte vorzusehen, sie zum Angebot eines Services als Provider autorisieren
- Services und Serviceoperationen sind also als "Ressourcen" in einem Rechtemodell zu berücksichtigen; die darauf möglichen Rechte sind "Consumer" und "Provider". Anders ausgedrückt sind pro Service bzw. pro Serviceoperation dedizierte Rollen vorzusehen, die zur Nutzung als Consumer und zum Angebot als Provider berechtigen.

Es kann praktisch jedes Rollenmodell genutzt werden, um Rechte individuellen Nutzern bzw. Systemen zuzuordnen. Im Fall von Service-Consumern ist beides möglich, im Falle von Provider ist nur die Autorisierung von Systemen sinnvoll.

Ein gute und etablierte Praxis ist es, neben individuellen Nutzern auch Nutzergruppen zu verwalten, die dann Rollen zugeordnet werden können. Die Rollen werden dann Rechten auf Ressourcen zugeordnet. Ein solches Modell erlaubt die effiziente Verwaltung von Rechten nicht nur im SOA-Kontext.

Parallel müssen geeignete User Interfaces zur Pfelge der Rollen, Rechte, Ressourcen, etc. bereitgestellt werden, die alle Aufgaben über den Lebenszyklus von Services (Anlegen, Mutieren, Entfernen) unterstützen müssen. Die konkreten Anforderungen funktionaler und nichtfunktionaler Art an solche Lösungen für das Access Management sind allerdings nicht Gegenstand dieser Referenzarchitektur.

Eine zu berücksichtende Alternative bzw. Erweiterung ist in 3.9.7 beschrieben.

3.9.7 Alternativ: Dezentrales IDM und Access Management durch Identity Federation

In der Praxis ist – speziell bei komplexen verteilten Anwendungen – ein vollständig zentrales Identity Management nur schwer machbar bzw. zu aufwendig. Dies ist insbesondere dann der Fall, wenn Nutzer aus verschiedenen Organisationen / Unternehmen beteiligt sind. Im konkreten Fall der Bundesverwaltung sind dies Nutzer aus der verschiedenen Departements, den Kantonen, anderen Verwaltungen und auch privaten Unternehmen; evtl. sogar Privatpersonen. Alle deren Identitäten zentral zu verwalten ist einerseits sehr aufwendig, andererseits oft unnötig wenn die Identität bereits in einer anderen Organisation gepflegt ist.
Ein Ansatz ist es, die Pflege von Identitäten in mehreren unabhängigen Identity-Managements Einheiten zu erlauben und die gegenseitige Weiterleitung von Identitäten zu erlauben. Man spricht dann von „Identity Federation“, wenn eine Organisation die Authentisierung durch eine andere vertrauenswürdige („trusted“) Instanz akzeptiert und also von dieser erzeugte Identitäten verarbeiten kann. Identitäten müssen dann nur an einer Stelle gepflegt werden, die Gewährung von Zugriffsrechten durch Dritte erfolgt im Vertrauen auf die Korrektheit der Authentisierung..
Für eine technische Umsetzung ist es notwendig, dass die beteiligten Organisationen Public Keys austauschen, so dass die von der einen Seite verschlüsselten (bzw. signierten) Tokens von der anderen Seite entschlüsselt (bzw. die Signatur geprüft) werden kann. Neben der eigentlichen Identität können auch Rollenzuordnungen oder Informationen über Berechtigungen im Token weitergegeben werden.

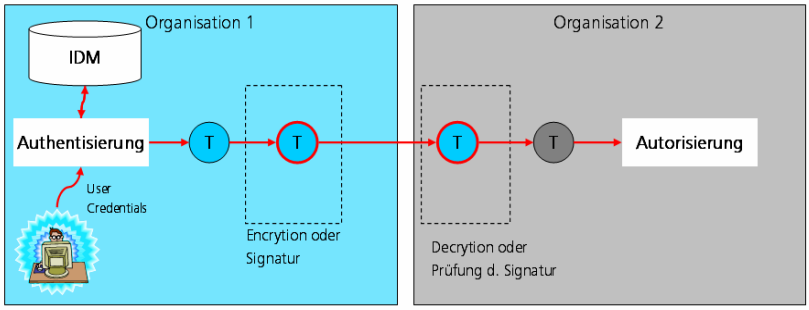

Abbildung 18: Schema für den Ablauf einer Identity Federation. Organisation 2 autorisiert auf der Basis einer in Organisation 1 authentisierten Identität.

Beispiel: Die Bundesverwaltung definiert ein Vertrauensverhältnis zu einer kantonalen Verwaltung bei der Nutzung von Services. Die kantonale Verwaltung kann nun ihren Nutzern die Berechtigung auf den Zugriff auf bestimmte Services des Bundes gewähren. Beim Zugriff dieser Nutzer auf die Bundesservices wird im Header ein Token mitgeschickt, dass die Identität und die Berechtigung codiert. Die Bundesverwaltung muss nur die Korrektheit des Tokens prüfen (etwa durch Prüfung der Signatur mit dem öffentlichen Schlüssel der kantonalen Verwaltung) und kann dann den Zugriff entsprechend der Berechtigung gewähren. Der Zugriff kann natürlich entsprechend vorher bilateral zu definierender Regeln eingeschränkt werden.
Auch das Access Managment, also die Gewährung von Zugriffsrechten auf der Basis von identifizierten Personen bzw. Rollen, kann verteilt sein. Dies erfordert in gewissem Sinne ein noch weitergehendes Vertrauensverhältnis der an der „Federation“ beteiligten Stellen, da eine Stelle direkt Rechte im fremden Verantwortungsbereich vergeben kann. Es ist daher in diesem Fall speziell wichtig, dass beim Aufsetzen der Identity Federation genau definiert wird, welche Rollen und welche Rechte föderiert werden sollen und welche nicht.
Identity Federation ist ein praktikabler Weg im eGovernment Umfeld. Die Basis dafür ist die Verwendung von standardisierten Tokens (etwa SAML oder Kerberos) für die Weiterleitung von Identitäten.

3.10 Übergreifende Patterns: Datensicherheit

In den bisherigen Abschnitten ging es um die Zugriffskontrolle, also um die Verhinderung von unautorisierten Zugriffen auf Services / Systeme. Für eine vollständige Sicherheit müssen aber auch die Inhalte der Messages berücksichtigt werden. Es sollen erreicht werden:

- Confidentiality: Verhinderung des Lesens von Nachrichteninhalten durch unautorisierte Personen / Systeme
- Integrity: Verhinderung der unautorisierten Manipulation von Nachrichten
- Authenticy: Verhinderung des missbräuchlichen Mehrfachversendung bzw. des Abfangens von Nachrichten durch Dritte (z.B.: Replay Attack)

Die Massnahmen hierzu sind:

- Signatur von Nachrichteninhalten (Manipulationssicherheit, eindeutige Kennzeichnung individueller Nachrichten)
- Verschlüsselung von Nachrichteninhalten

3.10.1 Sicherer Messagetransport (Transport-Level Security)

Es wird ein Protokoll zum Nachrichtentransport genutzt, das Mechanismen zur Datensicherheit unterstützt. Dies ist etwa mit HTTPS möglich, oder mit fast allen Queues (MOM).
Gegenüber unautorisierten Zugriffen auf den Transportkanal besteht so Sicherheit, am „Endpunkt" des Transports liegt die eigentliche Nachricht aber wieder im Ausgangszustand vor. Im Gegensatz zu „End-to-End Sicherheit" kann also immer nur ein „Ast" des gesamten Nachrichtentransports abgesichert werden.

3.10.2 Sichere Messages (Message-Level Security)

Hierunter wird auf der Ebene der Nachricht selbst eine Signatur und / oder Verschlüsselung genutzt, die unabhängig vom zum Versenden der Nachricht genutzten Transport funktioniert.
Bei Webservices wird etwa der „Payload" (die eigentlichen fachlichen Inhalte) verschlüsselt / signiert, wobei die notwendigen Steuerinformationen im Header plaziert werden. Dafür stehen mit SAML bzw. Kerberos auch etablierte Standards zur Verfügung.
Wichtig ist hier, dass nur der eigentliche Empfänger (ggf. eine bestimmte Person) einer Nachricht auf diese zugreifen kann, alle am Nachrichtentransport beteiligte Komponenten „in der Mitte" können das nicht. Dies kann ggf. Features wie „Content-Based Routing" (CBR) verunmöglichen.

3.10.3 PKI Infrastruktur für Schlüssel / Zertifikate

Falls Anforderungen an Datensicherheit gestellt werden, die Verschlüsselung oder Signatur voraussetzen, werden digitale Zertifikate benötigt.
Unter einer PKI (Public Key Infrastructure) versteht man ein System, dass solche digitalen Zertifikate ausstellen, verteilen und prüfen kann. Eine solche PKI ist notwendig, ihre genaue Funktionsweise ausserhalb des Scopes dieser Referenzarchitektur.

Beim Aufbau einer realen SOA-Plattform muss darauf geachtet werden, dass die Integration mit einer PKI effizient möglich ist. Bis zu einem gewissen Grad kann dabei die Unterstützung bestimmter Webservice-Standards (WS-Policy, WS-Security, WS-SecurityPolicy) helfen.

3.11 Registry und Repository

Je nach technischem Aufbau einer SOA-Plattform kann ein Registry unabdinbar sein und von Anfang an benötigt werden, oder auch nicht. Alternativ ist eine Konfiguration der verschiedenen Services im Service Bus möglich.
Ein Repository dagegen ist primär ein Tool für die Governance und wird erst in einer entwickelten SOA mit einer grösseren Zahl von Services wichtig.

Funktionale Lösungsmuster (Pattern):

- Registry:
    - Lookup von Services: Abfrage von Services auf Basis des eindeutigen Namens (meist eine URI), und Rückgabe des WSDL, der zugehörigen XML Schema Dokumente und anderer zur Laufzeit relavanter Artefakte (z.B.: WS-Policy Dokumente oder Konfigurationsfiles). Der

Service Bus, der in der Regel den Loookup durchführt, muss seine Konfiguration dynamisch auf Basis der Abfrage anpassen (siehe 2.5.1).
    - Publizieren von Services durch Einstellen der WSDL, XML Schema Dokumente und anderer Artefakte
- Repository:
    - Publikation von Informationen an das Registry: Wenn das Repository als Master für alle für Services relevanten Dokumenten dient, dann müssen die davon zur Laufzeit relevanten Dokumente an das Registry publiziert werden, sobald eine Änderung erfolgt ist. Damit bleibt das Registry stets aktuell.
    - Management von Design-Dokumenten aller Art: Neben der reinen Speicherung von Design-Dokumenten bieten Repositories auch Möglichkeiten zur Abildung von Abhängigkeiten zwischen Dokumenten sowie zur automatischen Prüfung der Konsistenz.
    - Suche / Recherche im Repository: Sicher nach Service-bezogenen Informationen und Artefakten; Navigation entlang der Abhängigkeiten und der logischen Struktur

Durch Policies zu regelnde Aspekte:

Im Rahmen der Vorgaben für Identifikation und Design von Services ist die Publikation in ein Repository bzw. ein öffentliches Verzeichnis zu regeln [1].

Die Nutzung einer Registry wird durch Vorgaben für das Deployment und den Betrieb von Services abgedeckt.