Bachelorarbeit

Konzipierung einer serviceorientierten Software-Architektur mit Hilfe einer ESB-Middleware für einen Profi-Fußballverein

Vorgelegt am:	16.09.2013
Von:	**Holger Lösken**
	Gustav-Adolf-Straße 41
	09116 Chemnitz

Studiengang:	Wirtschaftsinformatik
Studienrichtung:	Wirtschaftsinformatik
Seminargruppe:	WI10
Matrikelnummer:	4000746
Praxispartner:	das MedienKombinat GmbH
	Stefan-Heym-Platz 1
	09111 Chemnitz

Gutachter:	Herr Dipl.-Inf. René Nitzsche (das MedienKombinat)
	Herr Dr.-Ing. Wolfgang Oehme (Staatliche Studienakademie Glauchau)

Inhaltsverzeichnis

Inhaltsverzeichnis II
Abbildungsverzeichnis IV
Abkürzungsverzeichnis V
1 Einleitung 1
1.1 Problemstellung 1
1.2 Ziel 2
2 Das SOA-Modell 3
2.1 Einleitung 3
2.2 Das SOA-Modell 4
2.2.1 Die Architektur 4
2.2.2 Eigenschaften des SOA-Modells 6
2.2.3 Die wichtigsten Ebenen des Modells 6
2.2.3.1 Presentation 6
2.2.3.2 Orchestration 7
2.2.3.3 Services 7
2.2.3.4 Integration Architecture 7
2.3 SOAbility 9
2.4 SOA-Governance 13
2.5 SOA-Referenzmodelle 14
2.5.1 Das OASIS Referenzmodell für SOA 15
2.5.2 Das SOA Meta Model des W3C 18
2.5.3 Das generische Unternehmen 22
2.5.3.1 GeneriCo 22
2.5.3.2 Echtzeit-Unternehmen 24
2.6 SOA als Softwarearchitekturstil 28
2.6.1 Software-Architekturstile 28
2.6.2 Relevanz für SOA 31
2.6.2.1 Relevanz für die Praxis 32
2.7 Zusammenfassung 32
3 Von der Theorie zur Praxis – Die Umsetzung 35
3.1 Einleitung 35
3.2 Webservices 35
3.3 Prozess der Kundenanlage 38

3.4 Die Komponenten 40
3.4.1 ESB – Enterprise Service Bus 41
4 Wirtschaftliche Aspekte 45
4.1 Vorteile einer SOA – Der Nutzen 45
4.2 Allgemeine Herausforderungen - Umsetzung und Einführung einer SOA ... 46
5 Schlusswort 50
Quellenverzeichnis 51
Ehrenwörtliche Erklärung 52

Abbildungsverzeichnis

Abbildung 1: Die Architektur des SOA-Modells (eigene Darstellung) 5
Abbildung 2: Die Schichten der SOAbility (eigene Darstellung)..... 9
Abbildung 3: Die Verknüpfung zwischen einem SOA-Referenzmodell und einer SOA-Implementierung (eigene Darstellung)..... 15
Abbildung 4: Abhängigkeiten der Service-Dynamik gemäß OASIS..... 16
Abbildung 5: Architekturmodelle des W3C 19
Abbildung 6: Das Service Oriented Model des W3C 21
Abbildung 7: Betriebliches System gemäß GeneriCo (eigene Darstellung)..... 23
Abbildung 8: Die SOA-Referenzarchitektur 24
Abbildung 9: Die Ebenen der RTE-Architektur 26
Abbildung 10: Web-Service Landkarte für ein RTE 26
Abbildung 11: Architekturstile 28
Abbildung 12: Protokollstack für Web-Services..... 35
Abbildung 13: Aufrufstack für Webservices mit XML und SOAP (eigene Darstellung) 36
Abbildung 14: Prozesse für Kundenanlage..... 40
Abbildung 15: Architektur ohne ESB..... 41
Abbildung 16: Architektur mit Mule und Connectoren..... 42
Abbildung 17: Anbindung der Systeme Magento, TYPO3 und SugarCRM an den ESB (eigene Darstellung)..... 43
Abbildung 18: Vereinfachte Darstellung einer losen Kopplung von Prozessen, Services und Backend-Systemen..... 45

Abkürzungsverzeichnis

BPEL	Business Process Execution Language
BPM	Business Process Management
CORBA	Common Object Request Broker Architecture
CRUD	Create, Read, Update, Delete
EAI	Enterprise Application Integration
ESB	Enterprise Service Bus
QoS	Quality of Service
SaaS	Software as a Service
SOA	Service-orientierte Architektur
SOAP	Simple Object Access Protocol
SOP	Service-orientierte Plattform
WSDL	Web Service Description Language
URI	Uniform Resource Identifier

1 Einleitung

1.1 Problemstellung

In vielen Unternehmen wächst der Abstand zwischen der IT und den Fachbereichen, die sie unterstützen soll. Etablierte IT-Infrastrukturen ließen sich nur ungenügend an die sich immer rascher verändernden Anforderungen des Geschäftes anpassen. Ziel der IT eines Unternehmens war und ist es, Geschäftsprozesse zu unterstützen und das Unternehmen wettbewerbsfähiger und wirtschaftlicher zu gestalten. Die unzureichende Anpassungsgeschwindigkeit der IT führte in den letzten Jahren zu Unzufriedenheit und so vermehrt zu einer Verlagerung der Arbeit (Outsourcing, SaaS, externe Services), ohne das Problem an der Wurzel zu lösen.

Das Kernproblem besteht in dem hohen Aufwand, der entsteht, wenn komplexe existierende IT-Systeme an veränderte Geschäftsprozesse anzupassen sind. Notwendige Anpassungsfähigkeit verbunden mit kurzen Time-to-Market-Zyklen ist nichts Neues. Was sich ändert, ist die zunehmende Geschwindigkeit der Änderungen. Davon sind nicht nur unternehmensinterne Geschäftsprozesse betroffen, sondern zunehmend auch Geschäftsabläufe über Unternehmensgrenzen hinweg. Partnerunternehmen werden stärker in die eigene IT integriert und Outsourcing von IT-Dienstleistungen wird in einer serviceorientierten Architektur attraktiver. Durch diese Entwicklung bieten sich gerade für mittelständische Unternehmen völlig neue Marktchancen. Eine nachhaltige Problemlösung muss folglich an diesem Punkt ansetzen.

SOA, richtig eingesetzt, löst das Problem der unflexiblen IT grundlegender und nachhaltiger. Service-orientierte Architekturen stellen zeitgemäße Konzepte zur Verfügung, um IT-Infrastrukturen schrittweise flexibel und anpassbar zu gestalten. Gleichzeitig werden Geschäftsprozesse und das Management der Fachbereiche direkter verbunden.

Mit diesem Thema als Hintergrund ist in Abstimmung mit einem lokalen Profi-Fußballverein ein Konzept zur Lösung entstanden. Die Partnerschaft zwischen der Firma *das Medienkombinat GmbH* und dem Profi-Fußballverein besteht schon seit langem, sodass Abläufe und Prozesse bekannt sind. Aktivitäten seitens des Fußballvereins bei denen über Jahre hinweg immer ein großer IT-Datenpflegeaufwand entsteht, sollen nun durch einheitliche Schnittstellen, Synchronisation von Daten in verschiedenste Systeme und zentraler Ablage von Daten zukünftig vermieden werden. Dabei geht es auch darum, nicht nur die aktuellen Aktivitäten und Prozesse zu betrachten, sondern ein System zu schaffen, welches auch zukünftigen Anforderungen gerecht wird und nicht nach wenigen Jahren wieder komplett neu entworfen werden muss.

1.2 Ziel

In der vorliegenden Bachelorarbeit geht es darum, einen detaillierten Einblick in das Thema SOA zu erhalten, die Idee und ein Modell hinter dem Begriff verstehen zu lernen und die Umsetzung des Modells anhand von Anforderungen für einen Profi-Fußballverein nachzuvollziehen. Dabei werden nicht nur die einzelnen Insellösungen einen SOA-Kontext gebracht und verknüpft. Es wird als Basis eine ESB-Middleware eingesetzt um einen ganzheitlich flexiblen Ansatz zu gewinnen.

2 Das SOA-Modell

Dieses Kapitel gibt eine Übersicht über verschiedene SOA-Standardmodelle und die Vorstellungen der großen Hersteller. Es zeigt, wie ein herstellerunabhängiges und pragmatisches SOA-Modell aussieht.

Kapitel 2.2 beschreibt ein pragmatisches SOA-Modell, das die konkreten Stärken von SOA betont und die wichtigen Elemente einer SOA aufzeigt. Dieses Modell reduziert SOA auf das, was es wirklich ist: ein Mittel, um Zeit und Geld zu sparen.

Das Kapitel 2.3 versucht mit dem Vorurteil aufzuräumen, dass SOA ein Architekturstil sei. Im Gegenteil, mit SOA können die meisten Architekturstile umgesetzt werden. Diese Flexibilität ist eine der Stärken von SOA.

2.1 Einleitung

Die SOA-Modelle vieler Hersteller, Analysten und Berater wirken umfangreich und kompliziert. Hinter vielen dieser Modelle steckt die Vorstellung, mit SOA sämtliche Schwierigkeiten, die nun mal mit dem Bau, dem Betrieb und dem Unterhalt von Informationssystemen verbunden sind, lösen zu können. Hinzukommt der unübersichtliche Wald von Standards, die in Zusammenhang mit der Basistechnologie von SOA- Web Services - entwickelt werden. Beides neigt dazu, die zentralen Vorteile von SOA zu verdecken und SOA als ein Modell erscheinen zu lassen, das komplex und mit sehr großen Investitionen verbunden ist und nur durch essenzielle Veränderungen der bestehenden IT-Systeme umgesetzt werden kann. Die wahren Stärken des SOA-Modells liegen jedoch in der einfachen Erweiterung der üblichen Schichtung von Anwendungen und Architekturen um eine Service- und eine Orchestrierungs-Ebene und in der Möglichkeit, mit Hilfe von SOA Ordnung in eine bestehende heterogene Systemlandschaft zu bringen. Versteht man beispielsweise den ordnenden Charakter von SOA durch die Formulierung einer Servicelandschaft, wird sofort klar, dass SOA keinesfalls auf einen bestimmten Architekturstil beschränkt ist und auch keinen Architekturstil darstellt. Eine weitere große Stärke von SOA ist die relative Ähnlichkeit der Modelle der verschiedenen großen Hersteller. Dies deutet darauf hin, dass es in Zukunft sehr viel einfacher sein wird, die Produkte oder Produktfamilien verschiedener Hersteller zu einem funktionierenden Ganzen zu kombinieren. Noch einfacher wird SOA auf Unternehmensebene, wenn man von einem pragmatischen Ansatz ausgeht, der hilft, Kosten durch die Weiterverwendung bestehender Systeme zu sparen und Flexibilität durch die Trennung der Business-Logik in statische Funktionalität und dynamische Abläufe und Regeln zu erhöhen.

2.2 Das SOA-Modell

Eine wichtige Grundvoraussetzung für die erfolgreiche Einführung einer serviceorientierten Architektur in einem Unternehmen ist die Formulierung eines eigenen herstellerunabhängigen SOA-Modells. Ein solches Modell muss die drei wichtigste Stärken von SOA abdecken, nämlich

- die Weiterverwendbarkeit bestehender Systeme vorsehen,
- die Modellierung von ausführbaren Prozessen und Regeln erlauben und
- die zentralen Standards SOAP, WSDL und BPEL

unterstützen. Zudem sollte es für zukünftige, weitergehende Standardisierungen offen und sowohl auf einzelne Lösungen als auch eine gesamte IT-Infrastruktur anwendbar sein. Das Architekturmodell sollte hinsichtlich der Technologie neutral formuliert werden und gleichzeitig den Einsatz einzelner Produkte oder der gesamten SOA-Produktpalette eines oder mehrerer Hersteller gestatten. Diese Rahmenbedingungen bedeuten zwingend die Reduktion auf das Notwendigste und die Berücksichtigung eines gewissen Pragmatismus, was die Integration bestehender Anwendungen betrifft.

2.2.1 Die Architektur

Das SOA-Modell besteht aus folgenden Ebenen:

- *Presentation*: Das User Interface einer bestimmten Anwendung.

- *Orchestration*: Der Ablauf einer Anwendung wird als Workflow festgelegt. Dieser besteht aus einer Sequenz aller aufgerufenen Services, die gleichzeitig den Datenfluss zwischen diesen Services steuert.

- *Services*: In dieser Ebene sind alle Dienste enthalten.

- *Integration Architecture*: Die Infrastruktur zur Verbindung der verschiedenen Dienste und zur Verbindung von Diensten mit bestehenden Anwendungen/Systemen oder Datenbanken.

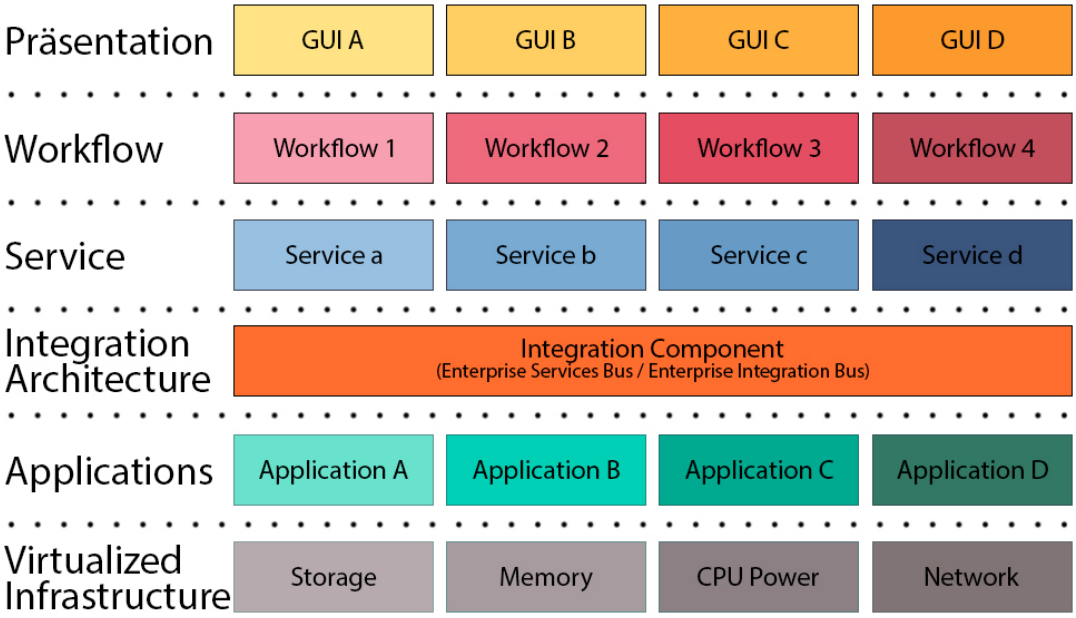

Abbildung 1: Die Architektur des SOA-Modells (eigene Darstellung)

- *Applications*: Diese Ebene besteht aus den neuen oder bereits vorhandenen Anwendungen – respektive deren funktionalen Blöcken – sowie aus Datenbanken oder anderen Datenquellen eines Unternehmens.
- *Virtualized Infrastructure*: Diese Ebene umfasst sämtliche zum Betrieb aller Komponenten des SOA-Modells notwendigen Ressourcen.

Das SOA-Modell betrachtet SOA als ganzheitlichen Ansatz einer kompletten IT-Systemlandschaft, die unterstützend für betriebliche Prozesse bereitsteht. Funktionen, die durch einzelne Systeme abgedeckt werden, sind durch SOA unternehmensweit in einer standardisierten Form zugänglich.

Das SOA-Modell sieht eine logische Teilung zwischen Anwendungen, Integration, Diensten und der Orchestrierung vor. Die Anwendungen (Applications) beinhalten bestehende oder auch gänzlich neue Systeme und logische Datenspeicher.

Die Dienste (Services) stellen Schnittstellen zu den einzelnen Anwendungen dar. Die Orchestrierung (Orchestration) dient zur Steuerung von Abläufen, die mittels Einbezug mehrerer Dienste durchgeführt werden. Die Kommunikation der Dienste untereinander und deren anwendungsbezogene Umsetzung (Anwendung und Daten) erfolgt über eine Integrationsarchitektur (Integration Architecture).[1]

Die Wiederverwendung bestehender Systeme ist ein integraler Bestandteil von SOA. Das bedeutet auch, dass ein strukturierter Umgang mit Legacy-Systemen notwendig ist, um deren Funktionalität optimal nutzen zu können. Oft bleibt es jedoch nicht bei der reinen Nutzung bestehender Systeme. Wird ein solches System einmal mit einem Service Interface betrieben, so werden die Fehler sichtbar. Sofern das eintritt muss man Teile ersetzen oder erweitern oder auch das komplette System neu

[1] Vgl. (LIEBHART, 2007, S. 27)

aufsetzen. Wenn es sich um ein Legacy-System handelt, das naturgemäß über die Jahre einen großen Funktionsumfang angesammelt hat, hilft nur ein Re-Engineering basieren auf einer durchgehenden Methodik, um die Qualität der Resultate hochzuhalten. Ein Re-Engineering eines größeren Systems in Richtung SOA ist in jedem Falle mit einem gestaffelten, mehrstufigen Verfahren zu realisieren, das eine Gewichtung der Komplexität anhand des Schwierigkeitsgrades der Umstellung beinhaltet. Oftmals lohnt es sich nicht, die Teile eines Legacy-Systems weiter zu verwenden, die nur mit sehr großem Aufwand „servicefähig“ gemacht werden können.

2.2.2 Eigenschaften des SOA-Modells

Das SOA-Modell besitzt grundlegend in folgende Eigenschaften:

- Die Trennung der Business-Logik in dynamische (Prozesse und Regeln) und statische (anwendungsbezogene Umsetzung) Bereiche. Der dynamische Bereich wird getrennt in einer separaten Schicht (Orchestration) als Prozess oder als Regel modelliert und ausgeführt. Ein Prozess wird immer als ausführbarer Prozess mit BPEL umgesetzt. Eine Regel wird immer als ausführbare Regel umgesetzt. Der statische Bereich der Business-Logik wird als Service realisiert. Dieser Service wird in jedem Fall als Web Service umgesetzt.

- Die Integration Architecture ist eine logische Komponente, die nicht unbedingt durch einen Enterprise Service Bus abgedeckt sein muss. Die minimale Realisierung der Integration Architecture ist ein Netzwerk und die Unterstützung der Internet Protokollfamilien HTTP, DNS und TCP/IP.

- Die Weiterverwendung bestehender Anwendungen ist ein integraler Bestandteil der Architektur. Die Funktionalität der Anwendung wird als atomarer Service gekapselt. Der Zugriff oder die Steuerung dieses Service erfolgt ausschließlich über eine Web Service-Schnittstelle.

- Der Service Layer unterstützt die beiden Standards SOAP und WSDL. Alle anderen Standards wie beispielsweise UDDI sind nicht notwendig, um eine SOA auf Unternehmensebene zu betreiben.

2.2.3 Die wichtigsten Ebenen des Modells

Die für den Bau einer Lösung basierend auf SOA wichtigen Ebenen sind Presentation, Orchestration, Services und Integration Architecture. Die beiden anderen Ebenen, Applications und Virtualized Infrastructure sind durch die bestehenden Systeme einer Unternehmung gegeben.

2.2.3.1 Presentation

Das User Interface einer mittels SOA realisierten Anwendung wird entweder als Po1tal oder als Office Application oder jedoch als Client Application realisiert. Client

Applications können sowohl als Rieb Client wie als Web Client umgesetzt werden. Im weitesten Sinne kann man Client Applications auch als Servicenehmer bezeichnen.

2.2.3.2 Orchestration

Die Orchestration-Ebene bildet Geschäftsprozesse und Geschäftsregeln in einer Service Oriented Architecture ab. Sie ist für den dynamischen Bereich der Business-Logik einer auf SOA basierenden Anwendung zuständig. Dabei kann zwischen der Dynamik auf Prozessebene und der Dynamik auf der Ebene einzelner Prozessschritte unterschieden werden. Geschäftsprozesse bilden Abläufe ab, die sich aus einer Sequenz aufgerufener Services zusammensetzen, die gleichzeitig den Datenfluss zwischen Letzteren steuern. Die Instrumente für die Modellierung und Ausführung von Geschäftsprozessen sind BPM und BPEL. Sie stellen die Dynamik auf Prozessebene dar. Geschäftsregeln formulieren Steuerungsparameter für das Verhalten der Abläufe einer Anwendung. Diese Regeln werden mit Rule Engines modelliert und zur Laufzeit ausgeführt. Sie stellen den dynamischen Bereich der Business-Logik auf Ebene des einzelnen Prozessschrittes dar.

2.2.3.3 Services

In dieser Ebene sind die Mechanismen zur Verwaltung von Diensten, die standardisierten Serviceschnittstellen sowie die spezialisierten Dienste zu finden. Die Mechanismen zur Verwaltung von Diensten unterstützen den Betrieb einer SOA auf Unternehmensebene. Aus betrieblicher Sicht sind Verantwortlichkeiten, Klassifizierung, Lebenszyklen, Dienstplanung, Versionsverwaltung, Testszenarien, Service Level Agreements und Verrechnungsmöglichkeiten für Dienste wichtige Elemente des Service Managements. Die standardisierten Serviceschnittstellen sind das Herzstück einer SOA. Das Interface zu einem Service muss in jedem Fall eine Web Service-Schnittstelle sein, die mittels WSDL beschrieben ist. Werden andere Schnittstellentechniken oder Protokolle eingesetzt, geht ein wichtiger V01teil von SOA verloren, und der Einsatz der Instrumente für die Orchestrierung, die Interoperabilität und die Verwendung von Diensten wird stark eingeschränkt. Spezialisierte Dienste im Rahmen einer SOA sind diejenigen Services, die auf Unternehmensebene Funktionalität zur Verfügung stellen, die weder einen bestimmten Fachbereich noch eine bestimmte Anwendung betreffen. Typische Vertreter sind Printing Services für ein unternehmensweites Output Management, Conversion Services für die Umwandlung von Formaten sowie Transformation Services für die Anreicherung respektive die Filterung von Daten.

2.2.3.4 Integration Architecture

Diese Ebene enthält die Infrastruktur zur Verknüpfung der verschiedenen Dienste und zur Verbindung von Diensten mit bestehenden Anwendungen oder Datenbanken sowie zur Koppelung von Services mit den Bestandteilen der Presentation-Ebene. Die entsprechenden Mechanismen können entweder als logische Integration, als Euterprise Service Bus (ESB) oder als Datenintegration realisiert werden. Die logische Integration ist die minimale Lösung, die lediglich ein HTTP Stack (DNS, TCP/IP, etc.) als Infrastruktur voraussetzt. Für die Realisierung vieler auf SOA basierender Lösungen reicht das völlig aus. Der Einsatz eines ESB oder einer Data Integration Plattform als Integration Architecture ist vor allem für größere SOA-

Realisierungen sinnvoll. Ein ESB stellt die Kommunikation zwischen verschiedenen Services sicher, übernimmt das Routing von Meldungen, erlaubt die Virtualisierung der Services und kann proprietäre Service-Schnittstellen als Web Service-Schnittstellen darstellen. Außerdem stellt jeder ESB weitere nützliche Instrumente zur Verfügung. Eine Data Integration Plattform ist ein spezialisierter ESB, der die unternehmensweite Harmonisierung und Konsolidierung von Daten unterstützt.

2.3 SOAbility

Das Kunstwort SOAbility, aus den Wörtern SOA und Ability, beschreibt die Fähigkeit einer bestehenden IT-Umgebung, das Konzept einer Service Oriented Architecture zu unterstützen. Dabei sind speziell zu betrachten:

- Applikationen, welche z.Z. in der Organisation genutzt werden.
- Infrastruktur, auf der diese Applikationen ablaufen.
- Organisation, welche Infrastruktur und Applikationen unterstützt.
- Technologische und organisatorische Umgebung.

Im Grunde sollten stets alle Aspekte der Organisation und der Softwaresysteme simultan betrachtet werden, um eine SOA-Migration einzuschätzen. Dies ist aber in der Regel nicht praktikabel. Es ist besser die SOAbility aus einer Schichtenperspektive (Abbildung 2) heraus zu betrachten. Jede Schicht hat ihre eigenen Charakteristika, nach der die partielle SOAbility beurteilt werden kann, aber die Einführung einer SOA macht nur dann Sinn, wenn alle Schichten die nötige SOAbility entwickelt haben. Insofern kann eine solche SOAbility-Untersuchung auch Defizite für die Einführung einer SOA offen legen.

Abbildung 2: Die Schichten der SOAbility (eigene Darstellung)

- *Applikationsschicht* – Die Applikationen sind der wichtigste Teil der Softwarestruktur einer Organisation. In einer Organisation, die noch nicht serviceorientiert arbeitet, kapseln die bestehenden Applikationen die bereits vorhandene Funktionalität, welche in Form von Services offengelegt werden muss. Diese vorhandenen Applikationen enthalten oft eine erstaunlich große Menge an mehr oder minder unbekannter Geschäftsprozessfunktionalität, die außerhalb der Organisation nicht oder nur sehr schwer zu finden ist. Mit zunehmender Serviceorientierung werden immer mehr und mehr Funktionen im Gesamtsystem als Services zugänglich gemacht.

Aber da ein Service anders strukturiert ist als eine traditionelle Applikation, können nicht alle Funktionen in einer Applikation auch als Service fungieren. Eine SOA beruht darauf, dass die Implementation eines Services austauschbar ist und damit für den Consumer unsichtbar bleibt, aber oft bleibt die Betrachtung der Implementation die einzige Möglichkeit, technische Informationen über Services zu gewinnen und im Rahmen der Umstellung auf eine SOA sind diese Services stets Teile von Applikationen.

- Zunächst sollte innerhalb der Applikationsschicht eine Reihe von Servicekandidaten bestimmt werden. Diese bilden die möglichen Zielservices. Für jeden Servicekandidat muss ein sogenanntes Lösungsprofil erstellt werden. Dieses Lösungsprofil wird später durch ein Zugriffsprofil ergänzt, um so den „besten“ Service zu ermitteln. Damit ein Lösungsprofil einsetzbar ist, benötigt es Aussagen über:
  - Wiederverwendbarkeit,
  - Verfügbarkeit,
  - Skalierbarkeit,
  - Modularität,
  - Fachlichkeit.
- Das Zugriffsprofil bestimmt, wie nahe ein Servicekandidat einem Service kommt. Ein solches Serviceprofil sollte zumindest folgende Punkte enthalten: Koppelung und Zugänglichkeit der Funktionalität.
- Außerdem wird das Interface des Servicekandidaten bestimmt, dabei müssen mehr als die üblichen fachlichen Verhaltensparameter definiert werden, speziell die nichtfunktionalen Eigenschaften und Qualitäten spielen eine wichtige Rolle.
- Das Kommunikationsprofil des Servicekandidaten bestimmt einen Teil der Verwendung. So muss geklärt werden, ob der Servicekandidat transaktionsfähig ist oder asynchron genutzt werden kann.
- Ein erstelltes Performanzprofil zeigt die möglichen Lücken für zukünftige Services auf. Da die meisten Serviceimplementierungen de facto auf SOAP und XML basieren, liegt es nahe, einige für diese Protokolle kritischen Größen zu betrachten:
  - Datengröße plus Overhead durch XML-Tags,
  - Parsinggeschwindigkeitsverlust,
  - Performanzverlust durch Verifikation,
  - Ver- und Entschlüsselungszeiten.
- – Ein Sicherheitsprofil ist unabdingbar, wenn der zukünftige Service extern genutzt werden soll.
- Ein Wissensprofil über das fachliche Wissen bezüglich des Servicekandidaten ist notwendig.
- Ein Organisationsprofil, wie der Servicekandidat unterstützt werden
- kann. Auf organisatorischer Ebene muss mit sehr viel mehr Kollaboration gerechnet werden als es in bisherigen Architekturen üblich ist, folglich muss auch die „Reife“ der Organisation in Bezug auf

die Unterstützung und Weiterentwicklung von Services betrachtet werden.

- *Infrastrukturschicht* – Die Frage, wie gut eine bestehende Infrastruktur eine SOA unterstützt, wird oft nicht betrachtet, sie ist aber für den Betrieb und die gesamte Softwareentwicklung eine der Voraussetzungen für die Einführung von Services. Folgende Punkte sollten für eine Beurteilung der SOAbility einer Infrastruktur betrachtet werden:
    - Kommunikation – Die Feststellung, dass eine Organisation Services (häufig Webservices) einsetzt, führt nicht notwendigerweise zu einer SOA. Oft werden Servicetechnologien nur eingesetzt, um P2PNetzwerke aufzubauen, aber eine SOA benötigt mehr als die bloße Fähigkeit, ein vorgegebenes Kommunikationsprotokoll zu nutzen. Die eingesetzten Kommunikationsprotokolle und -mittel sind ein notwendiges aber kein hinreichendes Kriterium für eine hohe SOAbility. Damit eine SOA unterstützt werden kann, müssen Servicerequests und das Routing der Aufrufe in einer konsistenten Art und Weise unterstützt werden.
    - Discovery – Das Auffinden eines Services zur Lauf- und Designzeit ist in großen Organisationen auf Grund der Menge an Services eine Herausforderung. Bedingt durch die lose Koppelung innerhalb der SOA, sollte die Charakteristik des dynamischen Auffindens und Bindings eines Services von der Infrastruktur explizit unterstützt werden.
    - Sicherheit – Die Infrastruktur muss auch in der Lage sein, unterschiedlichste Sicherheitskonzepte zu unterstützen.
    - Komposition – Die Infrastruktur muss Elemente wie Workflow oder eine BPEL-Implementierung enthalten, welche die Komposition von Services aktiv unterstützt. Dabei kann es zu zusätzlichen Anforderungen, wie der Transaktionsunterstützung für zusammengesetzte Services, obwohl die Subservices nicht transaktionsgestützt sind, kommen. Die Existenz von rekursiven Kompositionen muss zusätzlich unterstützt werden.
    - Management – Die Services, das Routing, die Performanz und die Verfügbarkeit müssen auch in der Infrastruktur verwaltet und beobachtet werden können. Neben der reinen Werkzeugunterstützung müssen im Betrieb auch die entsprechenden organisatorischen Randbedingungen und Mechanismen vorhanden sein.

- *Umgebungsschicht* – Ohne eine permanente Beobachtung und Interpretation der Umgebung der Organisation ist eine SOA nicht sinnvoll einsetzbar.

- *Organisationsschicht* – Die Einführung einer SOA macht nur dann Sinn, wenn es in der gesamten Organisation zu einer Serviceorientierung kommt, folglich muss auch die Organisation als solche bei einer Betrachtung der SOAbility

herangezogen werden. Damit eine SOA unterstützt werden kann, sind eine Reihe von Attributen der Organisation im Umgang mit Services zu bewerten:

- Servicehandling – Das Servicehandling überdeckt vier Kategorien:
  - Design,
  - Implementierung,
  - Komposition,
  - Wiederverwendung.

  Diese Tätigkeiten müssen aktiv durch die Organisation unterstützt oder entsprechend der Serviceorientierung als Service von Providern genutzt werden.

- *Qualität* – Die Methoden zur Qualitätsmessung und Steuerung müssen auf Services abgestimmt werden. Außerdem muss bei dem einzelnen Service auch stets der Nutzen für den Geschäftsprozess nachgewiesen werden können. Ein weiterer Aspekt im Sinne der Qualität sind eventuelle Ownershipkonzepte:
  - Wem gehört der Service?
  - Wem die Implementierung des Services?

  Erfahrungsgemäß ist die Qualität der Services eng mit dem Vorhandensein einer sinnvollen IT-Governance verknüpft, da diese homogenisierend auf die Serviceimplementierungen wirkt.

- *Management* – Services sind keine rein technische Frage, sondern eine Managementaufgabe, da die gesamte Organisation auf die Serviceorientierung ausgerichtet werden muss, dazu zählen:
  - Strategie und Planung,
  - Stakeholder,
  - Governance und Managementprozesse,
  - Anreize und Ziele.

- *Kultur* – Eine SOA kann sich nicht in jeder beliebigen Organisationskultur entwickeln, denn eine erfolgreiche SOA setzt voraus, dass die Organisationskultur folgende Werte explizit schätzt und fördert:
  - Prozessdenken;
  - Kollaboration – Ohne dass Arbeitsteiligkeit und gegenseitiges Vertrauen Kulturbestandteile sind, muss eine SOA scheitern. Kollaboration bedeutet immer Teamarbeit, aber im Fall einer SOA stets abteilungs- und bereichsübergreifend.
  - Wiederverwendung – Wiederverwendung darf nicht als Diebstahl geistigen Eigentums gelten, sondern muss als die beste Form der Softwareentwicklung angesehen werden

2.4 SOA-Governance

Die Einführung eines neuen Softwaresystems in einer Organisation und auch organisationsübergreifend verläuft in drei Phasen:

- *Adoptionsphase* – Die Beteiligten entscheiden sich, ob und wie sie das neue System nutzen wollen. Die Implementierung und Nutzung eines neuen Systems und ganz besonders einer neuen Architektur impliziert einen hohen Grad an organisatorischer Veränderung, wobei zu beachten ist, dass speziell im Fall der SOA-Einführung der externe Druck immens ist, mit der Folge, dass die vermeintlichen Vorteile deutlicher wahrgenommen werden als eventuelle Risiken.

- *Nutzungsphase* – Organisatorische Veränderungen werden vorgenommen, um das System nutzen zu können und es geht in den täglichen Gebrauch über.

- *Impactphase* – Während der Impactphase erfahren die Beteiligten die Vor- und Nachteile des eingeführten Systems. Ein weit verbreiteter Fehler beim Einsatz einer Governance ist der Glaube, dass man zuerst die Governance definieren kann und anschließend eine SOA implementiert. Dieses Vorgehen produziert in der Regel keine sinnvollen Resultate, da eine Governance nur im Rahmen einer konkreten Implementierungsform angemessen ist. Governance muss die Regeln zur Nutzung vorgeben, diese sind jedoch viel zu eng mit der Implementierung, korrekterweise mit den Kompromissen in der Implementierung, verknüpft als das es möglich wäre, diese vorab festzulegen. Diese Beobachtung l¨auft auf die Trennung in eine Designtimegovernance und eine Runtimegovernance hinaus.

Eine der wichtigsten Strategien zum Aufbau von großen Systemen ist die Reduktion von Komplexität. Ein solches Vorgehen ist nicht immer möglich, speziell dann nicht, wenn das Kernsystem eine hohe irreduzible Komplexität besitzt. In diesen Fällen ist die kontinuierliche Evolution des Systems die bessere Strategie. Während dieser Evolution muss es zu einer permanenten Adaption des Systems kommen. Ziel jeder Governance muss es daher sein, eine solche Evolution explizit zu unterstützen und diese in „natürliche“ Bahnen zu lenken.

Eine erfolgreiche Governance einer organisationsweiten SOA muss einige Punkte berücksichtigen. Dazu zählen:

- *Architekturgovernance* – Hierbei wird die zukünftige und bestehende Architektur nach Kriterien wie Standardisierung, Komplexität, Skalierbarkeit und Topologie gesteuert oder weiterentwickelt.

- *Service Lifecycle Governance* – Die Lebenszyklen einzelner Services müssen alle sehr ähnliche Charakteristika besitzen, insofern müssen Entwicklung, Betreibung und Veränderungen von Services einem technischen wie auch organisatorischen und softwaremethodischen Rahmen unterliegen:
    - Design und Implementierungsgovernance – Der Entwurf und Bau von Services muss einer Reihe von Blueprints und Qualitätsmaßstäben unterliegen. Fragen nach Art und Weise der Wiederverwendung, Dokumentation, Einordnung in Ontologien und Taxonomien, aber auch Designrichtlinien und Codingstandards für die Services werden hier erörtert.
    - Runtimegovernance – Die zentrale Fragestellung der Runtimegovernance ist: Wie werden Services in den Betrieb aufgenommen und wieder entsorgt? Spezielle Probleme bilden hier Fragen der Interoperabilität als auch Abhängigkeiten und das Fehlermanagement.

    - *Changegovernance* – Veränderung auch von Services ist unausweichlich. Daher werden im Rahmen der Changegovernance die Mechanismen der Komposition und Konfiguration aller Teile des Systems betrachtet.

2.5 SOA-Referenzmodelle

Referenzmodelle zeigen eine idealisierte Sichtweise auf eine Architektur, wie das Beispiel in Abbildung 3 zeigt. Sie haben eine technologie- und herstellerunabhängige Sichtweise auf eine bestimmte Architektur zum Ziel. Im Zentrum der Betrachtung stehen Referenzarchitekturen und Anwendungsszenarien, die bestimmte Anforderungen und Ziele erfüllen sollen und eine Reihe von Standards, Spezifikationen, Profile und Protokolle umsetzen.

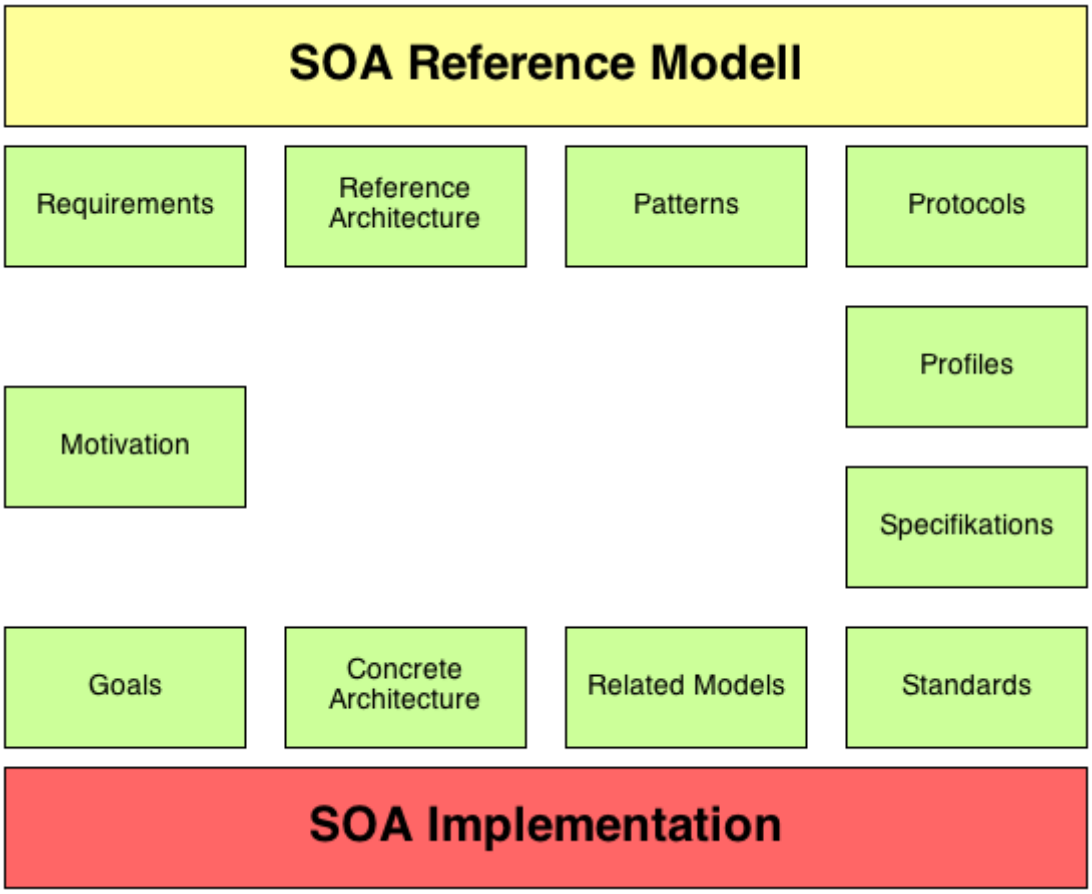

Abbildung 3: Die Verknüpfung zwischen einem SOA-Referenzmodell und einer SOA-Implementierung (eigene Darstellung)

Für die Praxis sind Referenzmodelle wichtig, da sie anhand einer Idealvorstellung die wichtigsten Konzepte hinter SOA konkretisieren. Außerdem werden diese Modelle von vielen großen Beratungshäusern und Analysten verwendet. Ein grundlegendes Verständnis der Referenzmodelle hilft, Beratungen von IT-Experten zu verstehen und anschließende das richtige Modell zu wählen. In diesem Kapitel werden die beiden Referenzmodelle der einflussreichsten Standardgremien (OASIS und W3C) und eine Auswahl von Modellarchitekturen skizziert:

- Das OASIS Reference Model for Service Oriented Architecture umfasst eine sehr genaue Beschreibung der Dynamik eines Service.
- Das SOA Meta Model des W3C formuliert ein detailliertes abstraktes Referenzmodell einer Global Web Service-Architektur, die auf den Prinzipien der WWW-Architektur basiert.
- Die generischen Modelle eines betrieblichen Informationssystems leiten aus einem abstrakten Betrieb eine generische SOA Modellarchitektur ab. Diese Modellarchitektur wird als Servicelandkarte dargestellt.

2.5.1 Das OASIS Referenzmodell für SOA

OASIS (Organization for the Advancement of Structured Information Standards) hat im August 2006 das “Reference Model for Service Oriented Architecture”

veröffentlicht. OASIS wurde 1993 gegründet und repräsentiert über 600 Organisationen in 100 Ländern. Das SOA-Referenzmodell soll als abstraktes Framework dienen, das ein Vokabular und eine Basis für das gemeinsame Verständnis bereitstellt.

OASIS versteht SOA als Paradigma für die Organisation und Nutzung verteilter Ressourcen, die dezentral verwaltet sein können. Services werden als Mechanismus verstanden, der Anforderungen und Ressourcen zusammenbringt. Das Referenzmodell konzentriert sich auf die Dynamik einzelner Services. Diese Dynamik basiert auf fünf wichtigen Konzepten. Sie beschreiben die Art und Weise, wie ein Service sichtbar ist, die Interaktion zwischen Services, die Art der Servicebeschreibung und des Ausführungskontextes, den Zweck einer Serviceinteraktion sowie den "Service Contract".

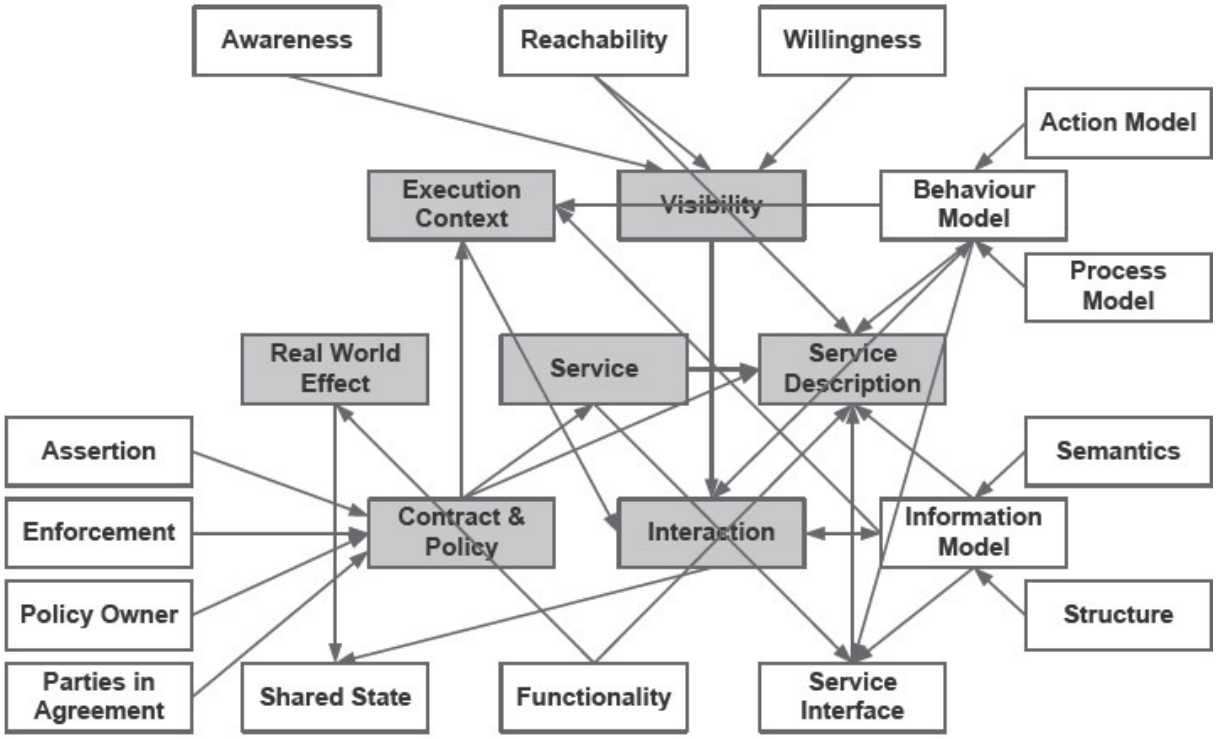

Abbildung 4: Abhängigkeiten der Service-Dynamik gemäß OASIS[2]

Die Dynamik eines Dienstes zu beschreiben, ist sehr schwierig. Die OASIS hat dies in ihrem Referenzmodell versucht. Abbildung 4 versucht, sämtliche Zusammenhänge aufzuzeigen, um die Komplexität dieser Dynamik zu illustrieren. Die wichtigsten Elemente der Dynamik (Visibility, Service Description, Interaction, Contract & Policy und Real World Effect) sind hervorgehoben.

- Visibility: Eine Interaktion zwischen Service Consumer (aufrufendem Prozess) und Service Provider (aufgerufenem Dienst) bedingt die Sichtbarkeit des Services, die an eine Reihe von Vorbedingungen geknüpft ist. Die Beteiligten - Consumer und Provider müssen von der gegenseitigen Existenz wissen

[2] (LIEBHART, 2007, S. 21)

(Awareness), miteinander interagieren wollen (Willingness) und einander erreichen können (Reachability).

- Service Description: OASIS bezeichnet die umfangreiche Servicebeschreibung als das Markenzeichen von SOA. Ziel der Beschreibung ist die Vereinfachung und Automatisierung der Interaktion und Sichtbarkeit von Services. Folgende Informationen sind wichtig:
    - Der Service existiert und ist erreichbar.
    - Der Service führt eine bestimmte Funktion oder bestimmte Funktionen aus.
    - Der Service funktioniert unter definierten Bedingungen und nach bestimmten Richtlinien.
    - Die Garantie, dass der aufgerufene Service entsprechend den Bedingungen und Richtlinien reagiert.
    - Die Beschreibung , welche Daten und Formate ein Service benötigt, um die erwarteten Aktionen auszuführen, die für eine bestimmte Funktionalität notwendig sind.

- Interaction: Die Interaktion zwischen Services erfolgt über ein gemeinsames Informationsmodell und ein so genanntes „Behaviour Model“, welches die Verhaltensweise des aufgerufenen Dienstes und die ggf. notwendigen Reaktionen des aufrufenden Prozesses beschreibt.

- Contract & Policy: Der „Service Contract“ beschreibt die Bedingungen für die Nutzung eines bestimmten Services, während die Richtlinien(Policy) den Rahmen dieser Bedingungen umfasst.

- Real World Effect: Die Interaktion mit einem Service verfolgt immer einen bestimmten Zweck. Dieser Zweck wird auch „Real World Effect“ genannt.

Relevanz für die Praxis

Das SOA-Referenzmodell von OASIS versteht sich als abstraktes und technologieunabhängiges Framework, welches dazu dient die wichtigen Entitäten und deren Beziehungen in einer SOA zu verstehen. Dies gelingt in Bezug auf Web-Services auch auf eindrückliche Art und Weise. SOA ist in den Augen von OASIS ein Paradigma für die Organisation und die Nutzung verteilter Ressourcen, die dezentral verwaltet sein können. Dreh- und Angelpunkt für die Interaktion zwischen den verteilten Ressourcen ist der Service. Die Stärke des Referenzmodells ist die klare Beschreibung der Dynamik des Zusammenspiels von Services. Es wird damit für die Realisierung einer SOA auf Unternehmensebene zu einer wertvollen Basisinformation. Die Schwäche des Referenzmodells ist das völlige Fehlen von Mechanismen wie beispielsweise der Orchestrierung, der Integrations-Architektur und anderer wichtiger Elemente einer SOA. Was geradezu störend wirkt, ist der fehlende Kontext zu den vielen OASIS-Standards für Web-Services. Insofern ist es nicht leicht, den Einsatz der Standards zu beurteilen.

2.5.2 Das SOA Meta Model des W3C

Die Web Services Architecture Working Group des World Wide Web Consortium (W3C) hat im Jahr 2004 eine so genannte Working Group Note veröffentlicht, die SOA als eine Ausprägung der Web Service Architecture beschreibt. Ausgangspunkt der Web Service Architecture ist die Definition des Web Service: „Ein Web Service ist ein Software-System, das konstruiert worden ist, um eine interoperable Maschinen-zu-Maschinen-Interaktion über ein Netzwerk zu unterstützen. Er hat eine Schnittstelle, die in einem für Maschinen verarbeitbaren Format beschrieben ist (WSDL- Web Service Description Language). Andere Systeme interagieren mit einem Web Service vorschriftsmäßig gemäß dessen Beschreibung über SOAP-Meldungen, die typischerweise über HITP mittels XML- Serialisierung in Verbindung mit anderen Web Service-Standards versendet werden."

Basis des SOA-Metamodells des W3C ist die WWW-Architektur, die als W3C Recommendation definiert ist.[3] Dieses Dokument beschreibt das Web als Informationsraum (Information Space) und dessen einzelne "Items of Interest" als Ressourcen, die über URI global identifiziert werden können.

Ein weiterer zentraler Begriff ist der Agent, der eine beteiligte Partei an einer Interaktion über das Web beschreibt. User sind "User Agents", Servicenehmer "Requestor Agents" und Services ,,Provider Agents".

[3] (JACOBS & WALSH, 2004)

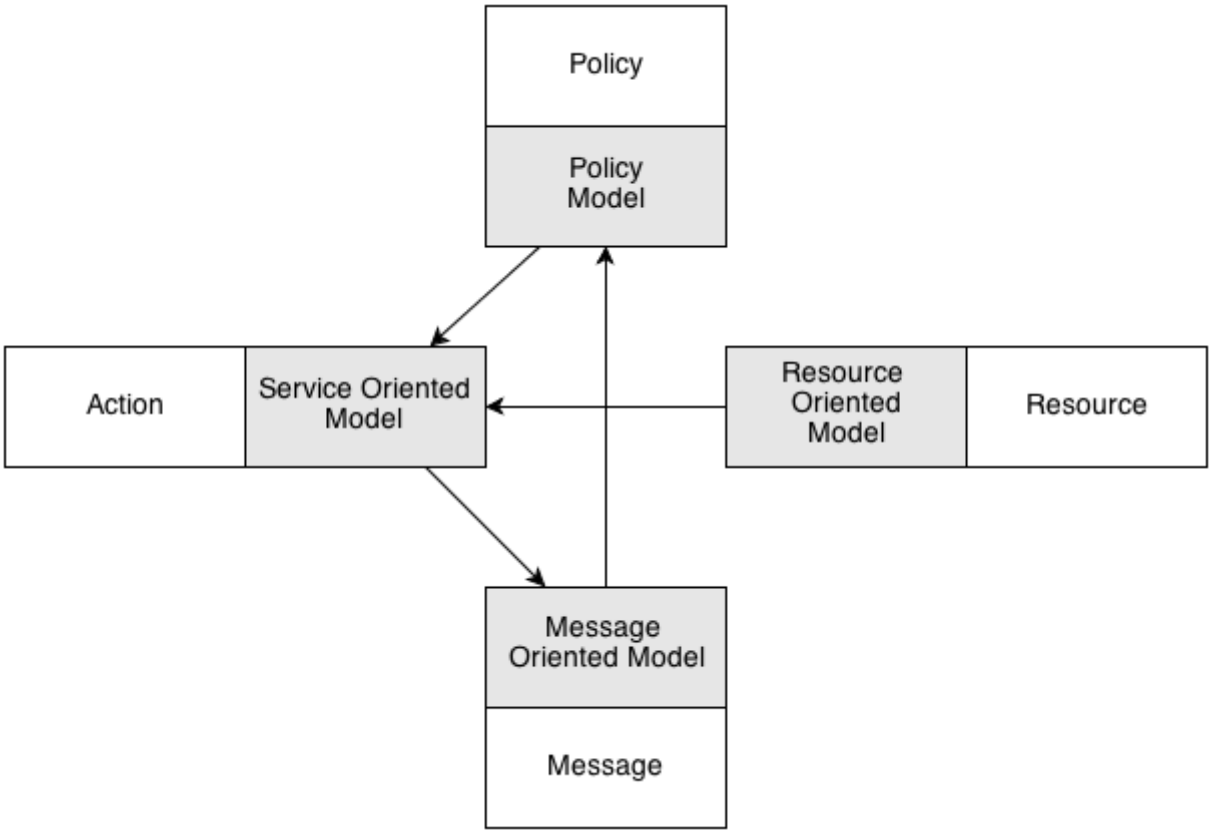

Abbildung 5: Architekturmodelle des W3C[4]

Das SOA Meta-Modell des W3C umfasst vier Modelle, die "teilweise geschichtet" miteinander korrelieren (Abbildung 5):

- *Message Oriented Model*: Dieses Modell beschreibt die Art und Weise, wie ein Agent die konkrete Implementation eines Service -Meldungen sendet und empfängt.

- *Resource Oriented Model*: Das Modell beschreibt die existierenden Ressourcen. Dieses Modell basiert auf dem der WWW-Architektur des W3C, das eine Ressource als Entität beschreibt, die eindeutig Identifiziert und mit der mittels definierter Formate interagiert werden kann.

- *Policy Model*: Das Policy Model definiert die für die Interaktion zwischen Ressourcen und Agents geltenden Grundsätze und Richtlinien. So werden beispielsweise Security und Quality of Service im Policy Model festgeschrieben.

- *Service Oriented Model*: Gemäß W3C ist dieses Modell eine Form eines verteilten Systems, das auf der Interaktion diverser getrennter Softwareagenten basiert. Die Eigenschaften dieses Modells sind folgende:

[4] (LIEBHART, 2007, S. 33)

- Logical View: Ein Service ist eine abstrahierte logische Sicht auf konkrete Programme, Datenbanken und Geschäftsprozesse. Diese Sicht definiert, was ein Service tut, typischerweise die Ausführung einer Operation auf Business-Ebene.

- Message Orientation: Der Service wird formell durch die Meldungen, die zwischen einem „Provider Agent“ und dem "Requestor Agent" ausgetauscht werden, definiert und nicht durch die Eigenschaften der Agenten selbst. Die konkrete Realisierung eines Service wird abstrahiert hinter der Service-Schnittstelle.

- Description Orientation: Ein Service wird durch maschinenlesbare Meta-Daten beschrieben. Die Semantik des Service wird ausschließlich durch seine Beschreibung dokumentiert.

- Granularity: Services tendieren dazu; relativ wenige Operationen werden mit komplexen Meldungen abgearbeitet.

- Network Orientation: Services werden dahingehend gestaltet, dass sie sich über ein Netzwerk ansprechen lassen.

- Platform Neutral: Die Meldungen werden in einem Plattform-unabhängigen und standardisierten Format übertragen.

Der Aufbau der vier Modelle und die beteiligten Objekte werden als Graph dargestellt. Sämtliche Objekte werden einzeln definiert.

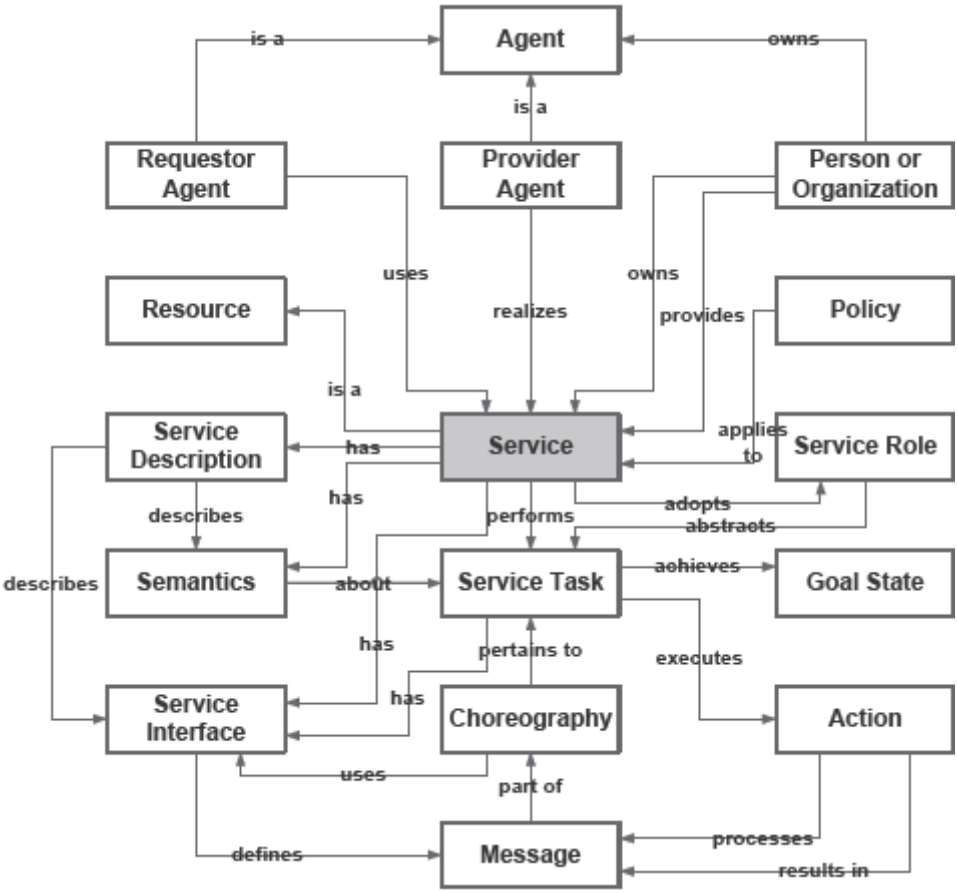

Abbildung 6: Das Service Oriented Model des W3C[5]

Das Service Oriented Model des W3C beschreibt, wie in Abbildung 6 dargestellt, die statischen Beziehungen zwischen den Objekten, die an der Realisierung eines Services gemäß W3C beteiligt sein sollten. Zentrale Konzepte sind der Service, die Aktion und der Agent. Der Service wird als Partei verstanden, die Aktionen auf Geheiß einer anderen Partei ausführt. Interessanterweise werden die konkreten Aktionen der Provider und der Requestor Agents aus der Betrachtung des W3C ausgeschlossen. Als Agent wird ein Software-Agent im Sinne der WWW Architecture Recommendation verstanden. Ein Agent realisiert immer einen Web-Service, er ruft auch immer einen Web-Service auf.

Relevanz für die Praxis

Das SOA Meta Model des W3C ist die Grundlage für das Verständnis der vielen Web-Service-Standards, weil das Modell auf der globalen WWW-Architektur basiert. Die WWW-Architektur ist wiederum eine wichtige Grundlage für das korrekte Funktionieren des Informationsraums Internet. Die Stärke des Modells liegt in der klaren und einfachen Darstellung der Beziehungen zwischen den zentralen Objekten, wie dem Service, der Message, der Resource und der Policy und anderen Entitäten

[5] (LIEBHART, 2007, S. 34)

wie beispielsweise der Aktion, der Serviceschnittstelle und vielen anderen mehr. Das W3C geht von einer globalen SOA- Architektur aus, also von einem globalen Markt der Funktionen. Darin bestehen auch die Schwächen des Modells in Bezug auf die Anwendbarkeit für ein Unternehmen. Die starke Betonung der Modelle in Richtung globaler Einsatzmöglichkeiten detailliert die einzelnen Entitäten so weit, dass eine konkrete Realisierung nur mit sehr großem Aufwand erfolgen kann. Dennoch ist und bleibt das SOA Meta Model des W3C für jedes Unternehmen ein wichtiges Instrument, um die verschiedenen Standards zu verstehen. Für die spezialisierten Gremien (Architektur, Compliance, Governance) größerer Unternehmen ist es ein absolutes Muss, dieses Modell zu kennen.

2.5.3 Das generische Unternehmen

Generische Unternehmensmodelle versuchen, aus einem sehr detaillierten, doch fiktiven Beispiel eine SOA-Referenzarchitektur zu entwickeln, die dann für ähnliche konkrete Unternehmen gelten soll.

2.5.3.1 GeneriCo

Im Jahr 2004 brachten 21 Firmen gemeinsam die SOA-Blueprints-Initiative in Gang, die versuchte, anhand einer SOA-Implementierung für das fiktive Unternehmen GeneriCo eine Referenzarchitektur zu realisieren. GeneriCo geht von einem betrieblichen Informationssystem für ein Unternehmen aus, das eine Reihe so genannter Enterprise Applications, einen allumfassenden Security-Mechanismus, ein Portal und eine Reihe von Individualanwendungen umfasst. Dieses System stellt Abbildung 6 dar.

Die Referenzarchitektur wurde abgestimmt auf eine Reihe von ausgewählten Funktionen:

- Ein verteilter , unternehmensweiter Sicherheitsmechanismus
- Ein Mitarbeiterportal
- Ein Katalog von Produktdaten mit Suchfunktion
- Eine Anwendung zur Personalverwaltung
- Ein Auswahl von Funktionen im Bereich der Gehaltsabrechnung

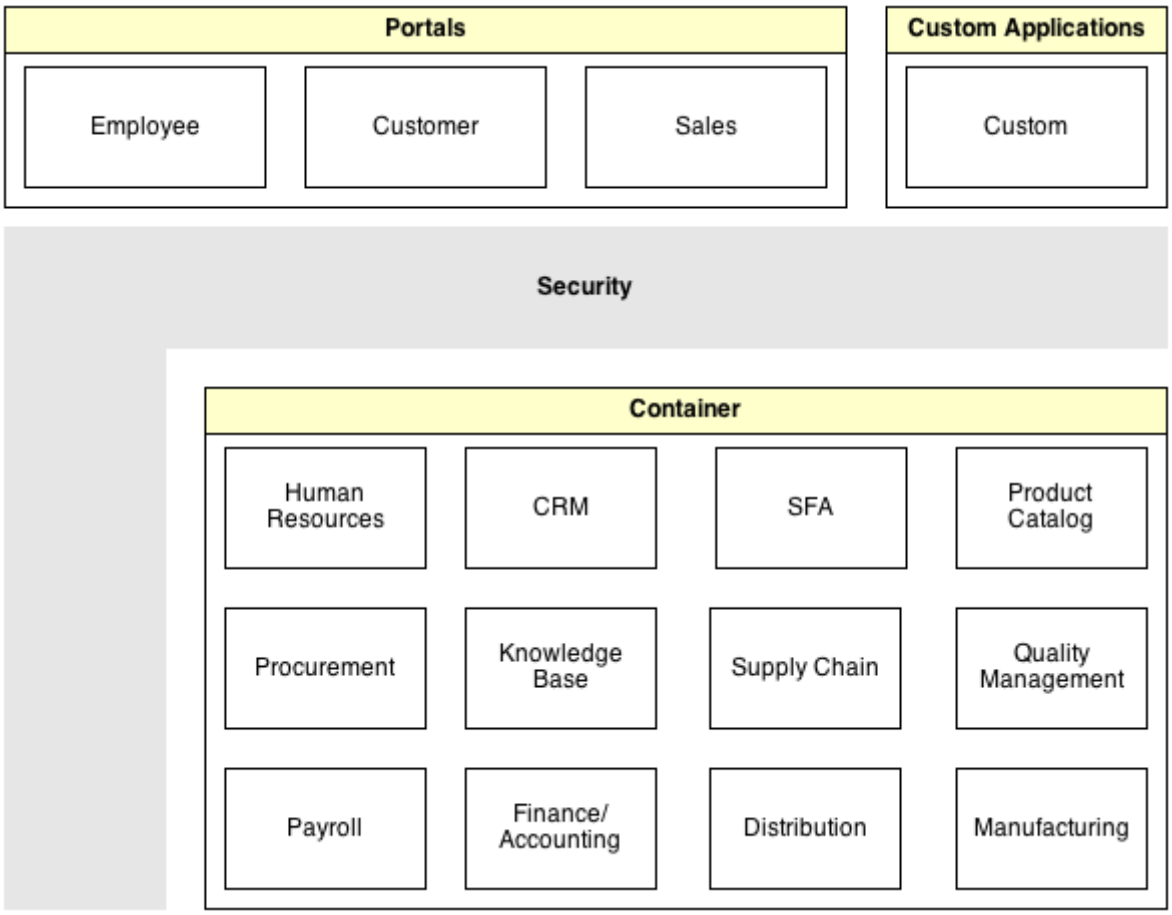

Abbildung 7: Betriebliches System gemäß GeneriCo (eigene Darstellung)

Die Servicelandschaft wurde im Rahmen der Spezifikation in verschiedene Bereiche aufgeteilt, wie Abbildung 7 zeigt. Aus dieser Spezifikation ergibt sich ein Service Layering, das Core Services, Enterprise Resource Services, Component Services, Data Services, Business Services und Workflow Services umfasst. Zudem sind Transportmechanismen und eine Reihe von Consumer Systems vorgesehen. Der Definition des Blueprints folgte ein so genanntes SOA Blueprints Reference Example, das vor allem Services und deren Interaktion für die Personalverwaltung und den Katalog der Produktdaten genau spezifiziert und die Realisierung des Portals für die Angestellten mit seinen Sicherheitsmechanismen erläutert.

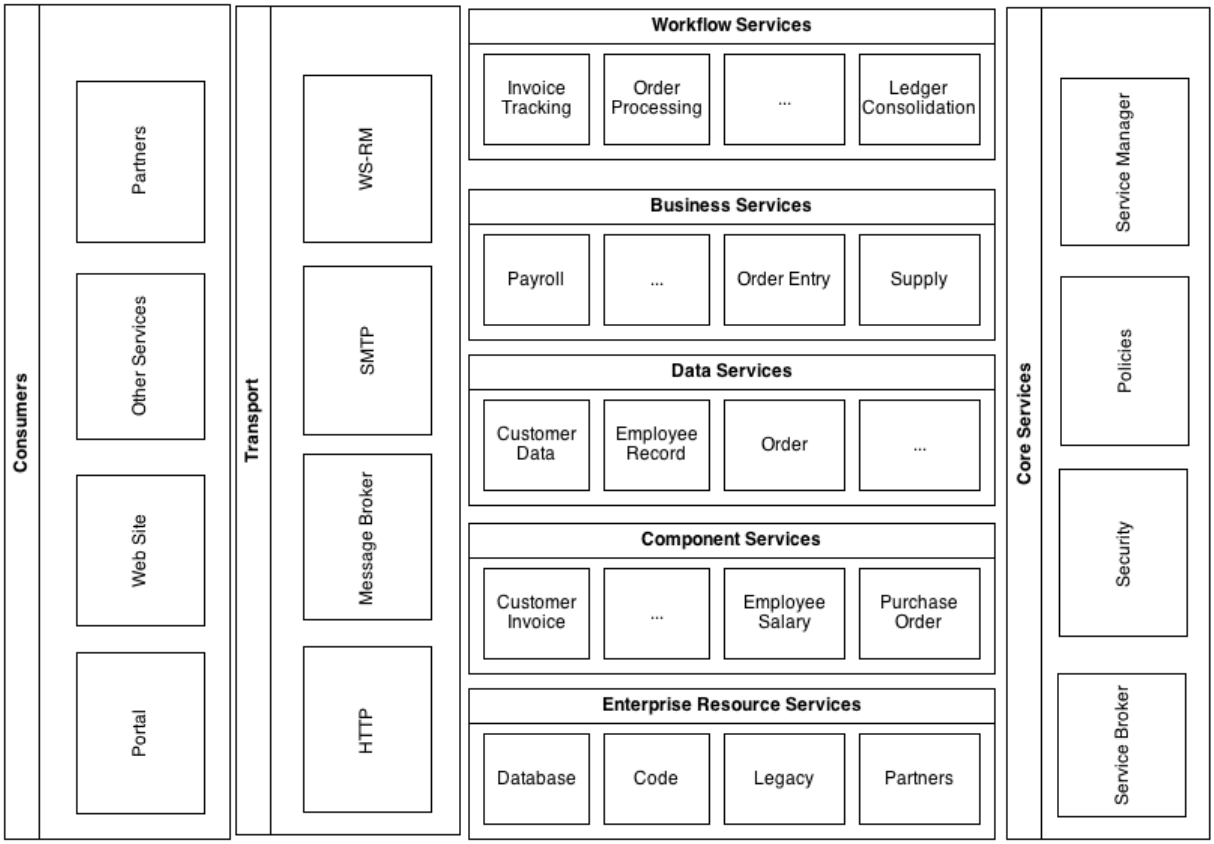

Abbildung 8: Die SOA-Referenzarchitektur

2.5.3.2 Echtzeit-Unternehmen

Ein wichtiges generisches Unternehmensmodell ist das so genannte Echtzeitunternehmen, welches sich aus den entsprechenden Modellen der ERP-Systemhersteller entwickelt hat. Echtzeit-Unternehmen oder auch Real-Time Enterprise (RTE) ist ein Unternehmensmodell, das davon ausgeht , dass sich ein Unternehmen in „Right-Time“ auf wechselnde Kundenbedürfnisse einstellen und die geforderten Produkte und Services am richtigen Ort zur richtigen Zeit liefern kann. Das Angebot und die intern erforderlichen Leistungsprozesse lassen sich sehr schnell an die Anforderungen der Kunden anpassen.

Real-Time Enterprise basiert auf einer Reihe von Prinzipien:

- *Anyhow*: Das Unternehmen liefert das Gewünschte mit allen Mitteln.
- *One-Stop*: Das Produkt oder der Service wird direkt ausgeliefert.
- *Everything*: Alles, was ein Kunde wünscht, wird geliefert.
- *Non-Stop*: Der Service eines Unternehmens steht jederzeit zur Verfügung.
- *Everywhere*: Der Service eines Unternehmens steht überall zur Verfügung.
- *One-To-One*: Es besteht eine direkte Kundenbeziehung.

Real-Time-Enterprise-Architektur

Damit Real-Time Enterprise als Unternehmensmodell umgesetzt werden kann, sind die betrieblichen Informationssysteme so zu gestalten, dass die „richtige Information zum richtigen Zeitpunkt am richtigen Ort zum richtigen Zweck verfügbar ist".

Die Umsetzung von RTE besteht aus Sicht der IT immer aus zwei grundlegenden Bestandteilen:

- *Flexible Prozesse*: Sämtliche Geschäftsprozesse sind vollständig modelliert und auf die entsprechenden IT-Systeme abgebildet. Eine Änderung am Geschäftsprozess zieht eine sofortige Änderung des unterstützenden IT-Systems nach sich. Diese Änderungen können in Echtzeit durchgeführt werden.
- *Real-Time Information*: Sämtliche Informationen eines Unternehmens sind immer auf dem aktuellen Stand und zeigen die momentane Situation des Unternehmens auf operativer, taktischer und strategischer Ebene.

SOA ist ein zentraler Bestandteil der Gesamtarchitektur eines RTE, der die beiden grundlegenden Eigenschaften eines solchen idealisierten Systems, Real-Time Information und flexible Prozesse, auf die Gesamtlandschaft der betrieblichen Informationssysteme eines Unternehmens abbildet. SOA ist damit eine Ebene der RTE-Architektur, wie in Abbildung 9 dargestellt.

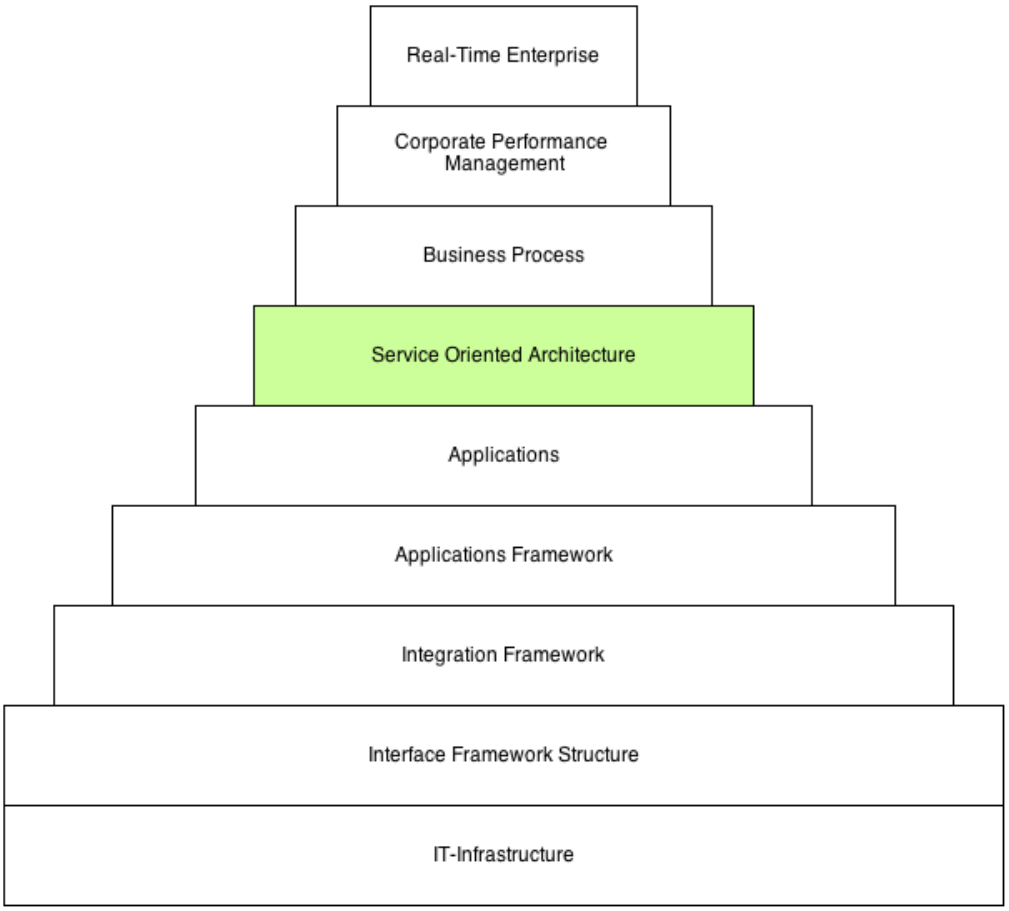

Abbildung 9: Die Ebenen der RTE-Architektur[6]

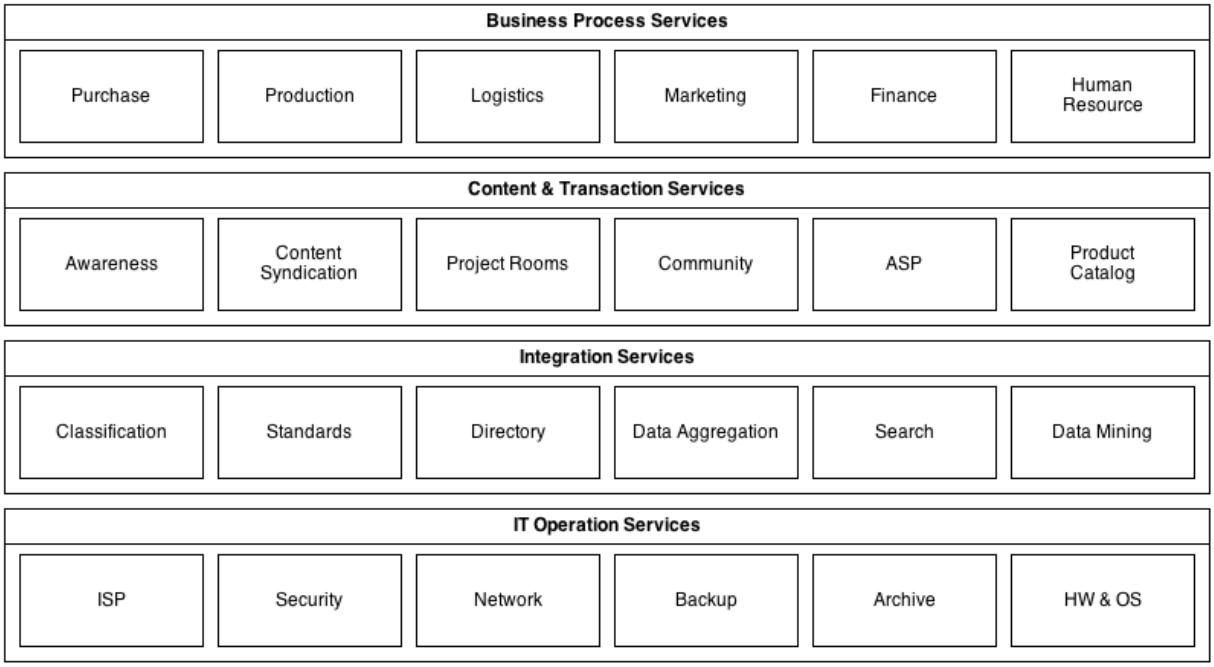

Abbildung 10: Web-Service Landkarte für ein RTE[7]

[6] (SCHMALE, 2004)

Die Servicelandkarte eines RTE hat vor allem Ordnungsfunktion. Es wird eine Zuordnung von Diensten auf verschiedenen Ebenen möglich. Obwohl die Landkarte ein idealisiertes Unternehmen darstellt, können solche Zuordnungen in der Praxis bei der Analyse bestehender Funktionalität sehr hilfreich sein. Die Landkarte, wie sie in Abbildung 10 dargestellt ist, sieht die Kategorien Business Process Services, Content & Transaction Services, Integration Services und IT Operation Services vor.

- *Business Process Services*: Die Services zur Unterstützung der Kernprozesse eines Unternehmens (Einkauf, Produktion, Logistik, Marketing, etc.) werden auf dieser Ebene realisiert.

- *Content & Transaction Services*: Die „Information Worker" Services, wie beispielsweise die Bereitstellung von internen und externen Informationen und die Unterstützung der Teamarbeit, werden durch Content & Transaction Services realisiert.

- *Integration Services:* Der Zugriff auf die verschiedenen Datenbereiche des Unternehmens erfolgt über diese Services.

- IT Operation Services: Die Basisdienste des IT-Betriebs.

Relevanz für die Praxis

Generische Unternehmensmodelle entwickeln aus einem sehr detaillierten, jedoch fiktiven Beispiel eine SOA-Referenzarchitektur. Alle diese Modelle gehen von einer gesamtheitlichen Betrachtungsweise des Unternehmens und seiner Kernprozesse aus. Sämtliche Prozesse werden durch eine SOA unterstützt. Viele Modelle - wie beispielsweise das GeneriCo-Modell der SOA-Blueprints-Initiative oder der Real-Time-Enterprise-Ansatz- resultieren in einer logischen Aufteilung der Services in verschiedene Bereiche, so genannte Service-Landkarten. Die Stärke dieser SOA-Referenzarchitekturen sind der illustrative Charakter der Lösung und die Gesamtsicht auf ein Unternehmen. Sie stellen daher eine wichtige Informationsquelle für ein Unternehmen der entsprechenden Branche dar. Die Schwäche dieser Modelle ist der fehlende Bezug zur Realität der bestehenden Systeme einer betrieblichen Informatik und somit die Illusion, dass SOA nur ein Ansatz für die „Enterprise Architecture" eines großen Unternehmens ist. Spätestens bei der Ausbreitung einer SOA in einem Unternehmen wird die Service-Landkarte zu einem wichtigen Struktur- und Kommunikationsinstrument.

[7] (LIEBHART, 2007, S. 38)

2.6 SOA als Softwarearchitekturstil

Die Basis von SOA ist die Verwendung von Services als fundamentalem Element für die Erstellung von Anwendungen. Hinter den Regeln der Kombination dieser Services verbirgt sich die Anwendung einer Reihe von klassischen Architekturstilen, deren Verständnis für den richtigen Einsatz von SOA wichtig ist. Architekturstile beschreiben Familien von Software-Systemen, die aufgrund ihrer Struktur und ihrer Semantik verwandt sind. Sie sind jedoch auch im Kontext einer bestimmten Zeit zu verstehen. In der groben Aufteilung Batch-Processing (60er Jahre), Call and Return (70er Jahre), Structured Decomposition (80er Jahre), Object Decomposition (90er Jahre) und Service Oriented Architecture (heute) sind verschiedene Architekturstile zu finden. Versuche, Architekturstile zu definieren und zu klassifizieren, fanden jedoch erst Ende der 90er Jahre statt[8], auch wenn die Definition etwas älter ist: "Falls Architektur ein formales Arrangement von architektonischen Elementen ist, wird ein architektonischer Stil dadurch definiert, dass er Elemente und formale Aspekte verschiedener spezifischer Architekturen abstrahiert. Ein architektonischer Stil ist weniger eingeschränkt und weniger vollständig als eine definierte Architektur."[9]

2.6.1 Software-Architekturstile

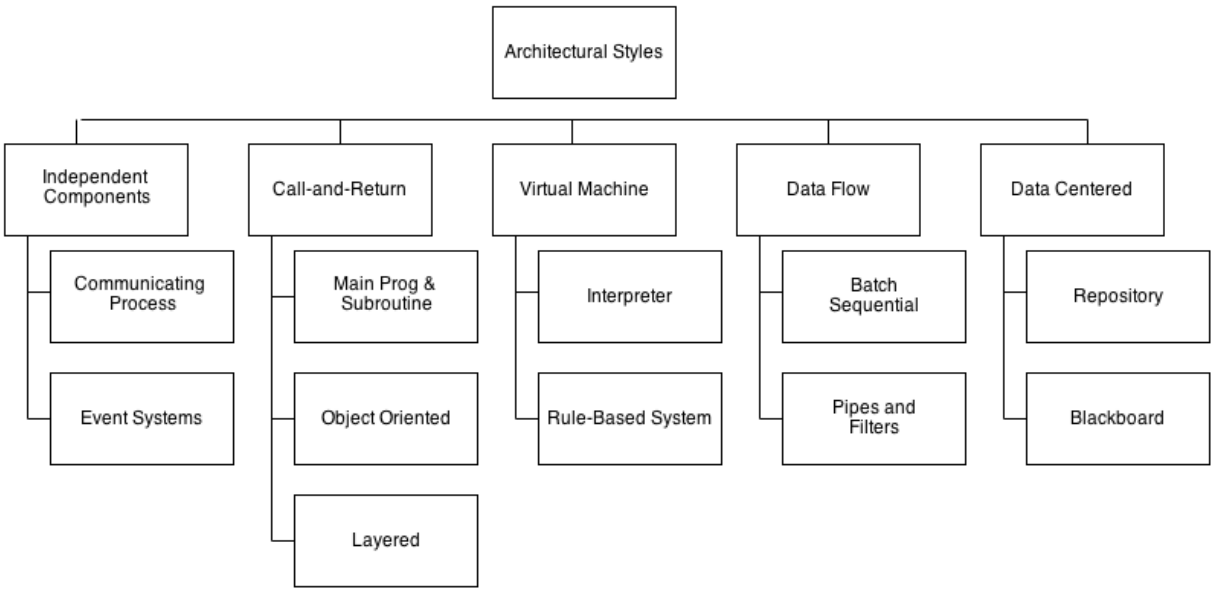

Abbildung 11: Architekturstile[10]

Software-Architekturstile können, wie in Abbildung 11 skizziert, in die Kategorien Independent Components, Call-and-Return, Virtual Machine, Data Flow und Data Centered eingeteilt werden.

[8] (MONROE, KOMPANEK, MELTON, & GARLAN, 1997)
[9] (PERRY & WOLF, 1992)
[10] (SHAW & GARLAN, 1996)

- *Independent Components*: Eine Architektur unabhängiger Komponenten besteht aus unabhängig ablaufenden Elementen, die über Nachrichten miteinander interagieren.

    - *Communicating Processes*: Dieser Architekturstil beschreibt aus einer beliebigen Anzahl miteinander kommunizierender Elemente bestehende Architekturen. Die Synchronisation erfolgt ausschließlich über die Kommunikationskanäle zwischen den Elementen und kann sowohl synchron als auch asynchron erfolgen.

    - *Event Systems*: Dieser ereignisgesteuerte Architekturstil definiert Architekturen, deren Prozesse über Events miteinander kommunizieren. Es wird zwischen Implicit Invocation (sämtliche Events werden an einen so genannten Event Space gesendet, der dann die Verteilung der Events an die betroffenen Prozesse übernimmt) und Explicit Invocation (die Events werden direkt zwischen den einzelnen Prozessen ausgetauscht) unterschieden.

- *Call-and-Return*: Call-and-Return-Architekturen sind durch einen fixen Kommunikations-Mechanismus zwischen aufrufendem und aufgerufenem Element definiert.

    - *Main Programm & Subroutine*: Dieser Architekturstil repräsentiert das klassische Programmierparadigma der SASD (Structured Analysis-Structured Design)-Welt. Ein eindeutiger Kontrollfluss durchläuft einen Baum, bestehend aus Hauptprogrammen und Unterprogrammen, wobei die Kontrolle dem gerade ausführenden Teil des Systems übeltragen wird.

    - *Object Oriented*: Der Object Oriented Call-and-Return-Stil folgt den Prinzipien des Stils Main Program & Subroutine, also der Organisation des Kontrollflusses über die jeweils aufgerufene Methode. Die Zugriffe auf Objekte erfolgt über Schnittstellen. Fokus des Stils ist das Information Hiding, die Kapselung von Daten.

    - *Layered*: Sämtliche Elemente dieses Architekturstils werden einzelnen Schichten zugeordnet, die hierarchisch geordnet sind. Jede dieser Schichten erfüllt eine klar definierte Aufgabe. Die Interaktion zwischen einzelnen Schichten erfolgt über definierte Protokolle.

- *Virtual Machine*: Eine Architektur, die auf virtuellen Maschinen basiert, erlaubt die Realisierung portabler und interpretierbarer Systeme.

    - *Interpreter*: Dieser Architekturstil beschreibt eine abstrakte Maschine und definierte Elemente für deren inneren Aufbau.

  - *Rule-Based System*: Dieser Stil wird für die Architektur von regelbasierten Systemen verwendet, die eine Generalisierung der Interpreter darstellen. Zentrale Eigenschaft dieser Systeme ist die Trennung zwischen der Maschine an sich und den Regeln, die die Maschine beschreiben.

- *Data Flow*: Eine Datenfluss-Architektur definiert ein System als Abfolge von datenbezogenen Transformationen.

  - *Batch Sequential*: Die Transformationen auf den Daten erfolgen durch voneinander unabhängige Elemente dieses Architekturstils. Dies bedeutet, dass die Daten als Ganzes eingelesen werden, um dann transformiert und anschließend wieder als Ganzes abgelegt zu werden.

  - *Pipes and Filters*: Die Transformationen auf den Daten erfolgen lokal (Filters) mittels Streaming (Pipes). Dies bedeutet, dass ein eingehender Data Stream laufend in einen ausgehenden Data Stream umgewandelt wird.

- *Data Centered*: Eine datenzentrierte Architektur beschreibt Systeme, deren wesentliche Aufgabe der Zugriff und die Aktualisierung von Daten eines Repositories ist. Dabei wird zwischen einem passiven Repository (Datenbank) und einem aktiven (Blackboard System) unterschieden.

2.6.2 Relevanz für SOA

SOA unterstützt verschiedene Architekturstile, ist jedoch für sich gesehen kein Architekturstil, sondern eine Standardarchitektur. Mit den Elementen von SOA lassen sich praktisch alle Architekturstile realisieren.

Stil	**Anwendung**	**Vor-/Nachteil**	**Relevanz**
Independent Components			
Communicating Processes	Parallel-verarbeitung	- *Vorteil*: einfache Modellierung, Skalierbarkeit - *Nachteil*: Komplexität der einzelnen Elemente	keine
Event Systems	GUI's, Real Time Systems	- *Vorteil:* Unabhängigkeit der Elemente, Änderungs-freundlichkeit - *Nachteil:* non-deterministisches Verhalten der Elemente	Event Processing auf Ebene Integration Architecture
Call-and-Return			
Main Program & Subroutine	Structured Programming, Client-Server (RPC)	- *Vorteil:* definierter Kontrollfluss - *Nachteil:* Skalierbarkeit, Erweiterbarkeit	Request/Reply Mechanismen für Service Calls
Object Oriented	allgemeines Design, Client-Server	- *Vorteil:* universell - *Nachteil:* Komplexität, Anwendungsfreiheit	keine
Layered	Multi-Tier Architectures	- *Vorteil:* konzeptionelle Integrität, Lokalität der Änderungen - *Nachteil:* Performance, Komplexität	Gesamtaufbau SOA
Virtual Machine			
Interpreter	Prozessor- und Betriebs-system-simulation	- *Vorteil:* Portabilität, Flexibilität - *Nachteil:* Performance	für Rule und BPEL Engines
Rule-Based Systems	Experten-Systeme	- *Vorteil:* Flexibilität durch Regelwerk	für die Orchestrierung

		- *Nachteil:* Komplexität, Performance	
Data Flow			
Batch Sequential	Host Systeme	- *Vorteil:* Datensteuerung - *Nachteil:* Flexibilität, Interaktion	oft als Prozess in SOA abgebildet
Pipes and Filters	Software Converter, Compiler	- *Vorteil:* Flexibilität, Verteilung - *Nachteil:* Komplexität	für die Daten-transformation
Data Centered			
Repository	Stammdaten-verwaltung	- *Vorteil:* einfach - *Nachteil:* Single Point of Failure	zentrales SOA-Element
Blackboard	Kontroll-systeme	- *Vorteil:* skalierbar - *Nachteil:* Anwendung eingeschränkt	keine

2.6.2.1 Relevanz für die Praxis

In der Praxis führt die Definition von SOA als Architekturstil zu einer großen Verwirrung. Es entstehen leider unnötige Diskussionen, wenn man davon ausgeht, dass es eine richtige und eine falsche stilistische Umsetzung von SOA gibt. Diese Diskussionen blockieren die vernünftige Einführung einer SOA in einem Unternehmen. Man darf sich weder von Analysten, noch von Beratern oder Herstellern beeindrucken lassen. SOA ist und bleibt eine Standard-Architektur mit der sich alle wichtigen Architekturstile umsetzen lassen. Anderenfalls wäre das Einsatzgebiet einer SOA äußerst beschränkt und könnte niemals als Basis für die kostengünstige, flexible und standardisierte Realisierung von Anwendungen in einem Unternehmen dienen.

2.7 Zusammenfassung

Das SOA-Modell (vgl. Abbildung 1) versteht SOA als ganzheitliche Betrachtung einer IT-Systemlandschaft, die als Unterstützungsfunktion für betriebliche Prozesse realisiert wird. Von einzelnen Systemen abgedeckte Funktionen sind dank SOA in einer standardisierten Form unternehmensweit, ja sogar über die Unternehmensgrenzen hinaus zugänglich.

Das SOA-Modell umfasst die wichtigen Ebenen Presentation, Orchestration, Services und Integration Architecture sowie die Ebenen Applications und Virtualized Infrastructure:

- *Presentation*: Das User Interface einer bestimmten Applikation.

- *Orchestration*: Der Ablauf einer Applikation wird als Workflow festgelegt. Diese Abläufe bestehen aus einer Sequenz von aufgerufenen Services, die gleichzeitig den Datenfluss zwischen diesen Services steuert.

- *Services*: In dieser Ebene sind sämtliche Dienste enthalten.

- *Integration Architecture*: Die Infrastruktur zur Verbindung der verschiedenen Dienste und zur Verbindung von Diensten mit bestehenden Applikationen/Systemen oder Datenbanken.

- *Applications*: Diese Ebene besteht aus den neuen oder bereits vorhandenen Anwendungen - respektive deren funktionalen Blöcken - sowie aus Datenbanken oder anderen Datenquellen eines Unternehmens.

- *Virtualized Infrastructure*: Diese Ebene umfasst sämtliche, für den Betrieb aller Komponenten des SOA-Modells erforderlichen Ressourcen (Hardware, Netzwerk, Storage, etc.).

Die Stärken des SOA-Modells liegen in der einfachen Erweiterung der üblichen Schichtung von Anwendungen und Architekturen um eine Service-Ebene und eine Orchestrierungsebene und in der Möglichkeit, durch SOA Ordnung in eine bestehende heterogene Systemlandschaft zu bringen.

Wichtige Eigenschaften des SOA-Modells:

- Die Trennung der Business-Logik in dynamische (Prozesse und Regeln) und statische (applikatorische Umsetzung) Bereiche. Der dynamische Bereich wird getrennt davon in einer separaten Schicht (Orchestration) als Prozess oder als Regel modelliert und ausgeführt. Ein Prozess wird immer als ausführbarer Prozess mit BPEL umgesetzt. Eine Regel wird immer als ausführbare Regel umgesetzt. Der statische Bereich der Business-Logik wird als Service realisiert. Dieser Service wird in jedem Fall als Web Service realisiert.

- Die Integration Architecture ist eine logische Komponente, die nicht unbedingt durch einen Enterprise Service Bus abgedeckt werden muss. Die minimale Realisierung einer Integration Architecture ist ein Netzwerk.

- Die Weiterverwendung bestehender Anwendungen ist integraler Bestandteil der Architektur. Die Funktionalität dieser Anwendungen wird als Service gekapselt. Der Zugriff oder die Steuerung dieses Service erfolgt ausschließlich über eine Web Service-Schnittstelle.

- Alle großen Hersteller unterstützen mit ihren Produkten dasselbe SOA-Modell. Es wird also wesentlich einfacher werden, verschiedene Produkte verschiedener Hersteller zu einem sinnvollen Ganzen zu kombinieren.

- Der Service Layer unterstützt die beiden Standards SOAP und WSDL. Jeder andere Standard, wie beispielsweise UDDI, ist nicht notwendig, um eine SOA auf Unternehmensebene zu betreiben.

Für die Praxis sind Referenzmodelle wichtig, da sie anhand einer Idealvorstellung die wichtigsten Konzepte hinter SOA konkretisieren. Außerdem verwenden diese Modelle viele Analysten und große Beratungshäuser. Ein grundlegendes Verständnis der Referenzmodelle hilft, die Angebote der Beratungshäuser, der Analysten und auch der Consultingabteilung der großen Hersteller zu bewerten und das Richtige auszuwählen. Es existieren zwei Referenzmodelle der einflussreichsten Standardgremien (OASIS und W3C) sowie eine Reihe von Modellarchitekturen. Das OASIS Reference-Modell for Service Oriented Architecture umfasst eine sehr genaue Beschreibung der Dynamik eines Service. Das SOA Meta Model des W3C formuliert ein detailliertes, abstraktes Referenzmodell einer Global Web Service-Architektur, die auf den Prinzipien der WWW-Architektur basiert. Die generischen Modelle eines betrieblichen Informationssystems leiten aus einem abstrakten Betrieb eine generische SOA-Modellarchitektur ab. Diese Modellarchitektur wird als Servicelandkarte dargestellt.

Will ein Unternehmen SOA einsetzen, sind in jedem Fall bestehende Systeme und Infrastrukturkomponenten betroffen. Jedes Unternehmen hat eine Reihe von Standard-Produkten bestimmter Hersteller im Hause. Es gibt kein Unternehmen, das nicht Software einsetzt, die entweder von IBM, SAP, Oracle oder Microsoft stammt oder auf Open Source- Technologie basiert. Diese Hersteller und die spezialisierten Hersteller wie beispielsweise BEA gehen von ein und demselben Architekturmodell für SOA aus. IBM versteht SOA als Praxis, die darauf basiert, dass die Architektur eines Informationssystems aus der Geschäftsarchitektur, einer Zusammenstellung von Services eines Unternehmens sowie den entsprechenden Richtlinien und Regeln für deren Einsatz abgeleitet werden kann. SAP sieht SOA als Instrument zur Abbildung betrieblicher Prozesse in die Informationssysteme eines Unternehmens, also als so genannte Enterprise SOA, die auf Enterprise Web Services aufbaut. Für Oracle ist SOA eine Standard-basierte Plattform, die es erlaubt, Dienste (Web Services) bereitzustellen, aufzufinden und gegenseitig zu benutzen. Microsoft versteht SOA als Resultat bestimmter Grundsätze, Praktiken und Frameworks, die es erlauben, normierte Services bereitzustellen, sowie als Architektur einer Applikations-Plattform. Die Open Source-Gemeinde verfügt zwar über kein SOA-Modell, jedoch über eine große Zahl verschiedener Tools.

SOA unterstützt verschiedene Architekturstile (Event Systems, Call-and-Return, Rule-Based Systems, Batch Sequential, Pipes and Filters, Repository), ist jedoch für sich gesehen kein Architekturstil, sondern eine Standardarchitektur. Mit den Elementen von SOA lassen sich praktisch alle Architektur-Stile realisieren.

3 Von der Theorie zur Praxis – Die Umsetzung

3.1 Einleitung

In diesem Kapitel wird beschrieben, wie die Anbindung zwischen mehreren Systemen, die alle gänzlich unterschiedliche Aufgaben in einem Profi-Fußballverein erfüllen, über einen Enterprise Service Bus miteinander verknüpft werden. Dazu werden Prozesse definiert, die bei der täglichen Arbeit häufig vorkommen und mit Hilfe des SOA-Gedanken für die Zukunft effizienter durch die IT abgebildet und unterstützt werden sollen. Aufbauen auf dem Kapitel 2, welches das Konzept SOA im Detail betrachtet, wird in diesem Kapitel anfangs der Webservice an sich genau untersucht, da das Konzept gänzlich auf Web-Services aufbaut

3.2 Webservices

Webservices können als ein Produkt der Entwicklung des World Wide Webs (WWW) gesehen werden, da neben der reinen Darstellung und Verlinkung von Dokumenten auch der Informationsaustausch recht schnell in den Vordergrund des WWW rückte. Die ersten Versuche basierten auf einer Art Screenscrapertechnik, wobei das gesendete HTML interpretiert wurde. Die automatische Erkennung von HTML führte recht schnell in eine Sackgasse und die Softwareentwickler wandten sich XML zu, welches über HTTP transportiert wurde. Die Nutzung von Webservices führt nicht automatisch zu einer SOA, ganz im Gegenteil, frühe Webserviceimplementierungen haben die Tendenz, sich zu fest codierten Aufrufen von Services zu entwickeln, die einfach ein Webserviceprotokoll einsetzen. Genauso gut könnte auch direkter Call gemacht werden. SOA ist eine Architekturidee und Webservices sind ein bestimmtes Protokoll.

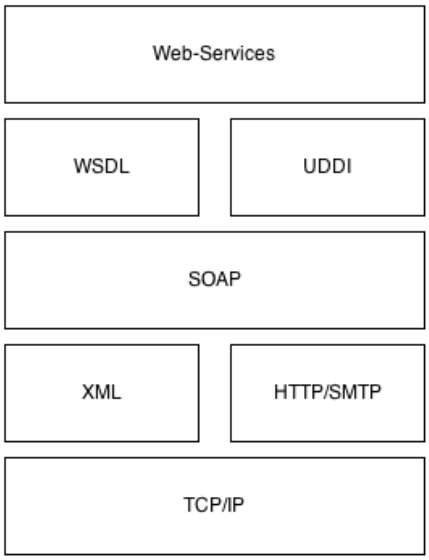

Abbildung 12: Protokollstack für Web-Services[11]

[11] (MASAK, 2007, S. 116)

Oft werden Webservices mit dem Konzept der mobilen Agenten verwechselt. Mobile Agenten werden üblicherweise eingesetzt, um auf einem anderen System Operationen durchzuführen. Sehr oft steckt dahinter die Idee des autonomen mobilen Agenten, welcher sich wie ein Wurm oder Virus völlig unabhängig in dem anderen System bewegt. Typisch für solche mobilen Agenten sind Aufgaben im Bereich der Informationsbeschaffung. Mobile Agenten können auch als Webservices implementiert sein, jedoch ist nicht jeder mobile Agent ein Webservice. Markante Unterschiede sind:

- Ein Web-Service hat nur Wissen über sich selbst, nicht jedoch Wissen über seine Consumer oder andere Webservices in seiner Umgebung. Im Gegensatz dazu sind mobile Agenten meistens damit beschäftigt, Informationen über ihre Umgebung und ihre Consumer zu sammeln. Diese Form der ”Self Awareness“ ist manchmal nur auf einer Metaebene vorhanden, trotzdem ist sie immanent.

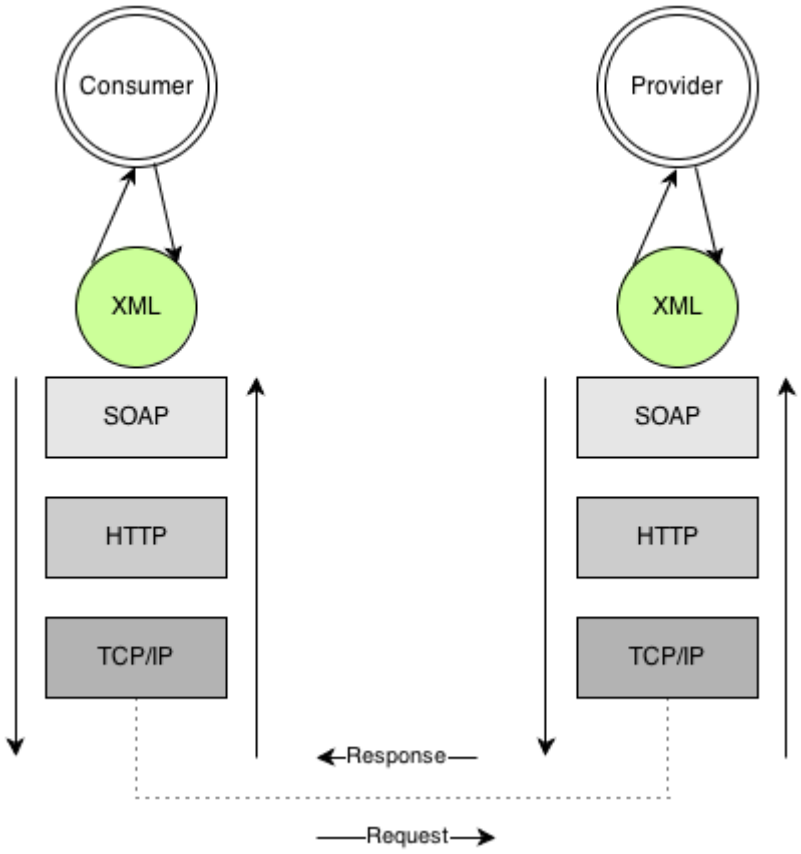

Abbildung 13: Aufrufstack für Webservices mit XML und SOAP (eigene Darstellung)

- Agenten streben von sich aus Aktivitäten an, Webservices sind stets passiv.
- Agenten sind bis zu einem gewissen Grad autonom, Webservices sind dies heute nicht, obwohl die Autonomie eine zentrale Forderung an die Services ist.

- Agenten werden meist a priori kooperativ konzipiert, während dies Webservicestandards heute nicht berücksichtigen.
- Agenten müssen sich auf diverse Umgebungen einstellen können, bei Webservices ist diese Umgebung heute faktisch fixiert.

Obwohl komponentenbasierte Architekturen schon ein gewisses Alter haben, ist das wirklich Neue an den Webservices ihre lose Koppelung, welche sich auch aus der SOA ableitet. Hierin unterscheiden sie sich drastisch von traditionellen Komponentenarchitekturen wie CORBA. Die zweite interessante Entwicklungslinie ist EAI. Die EAI-Systeme sind im Vergleich zu den Webservices teurer, da die Investitionsvoraussetzungen für eine EAI sehr viel höher sind als für die Webservices. Aus technischer Sicht erzeugt eine EAI keine flexiblen generischen Interfaces, die eine ideale Voraussetzung für Wiederverwendung darstellen. Mittelfristig könnte es zu einer gewissen Koexistenz zwischen beiden Techniken kommen, mit einer EAI innerhalb eines Subsystems und den Webservices innerhalb des Intra- und Internets. Auch die bestehenden monolithischen Legacyapplikationen können von den Webservices genutzt werden. In diesem Fall erhalten die Legacyapplikationen zusätzliche Interfaces, die Teile ihrer monolithischen Funktionalität als Webservices zur Verfügung stellen. Dieses Vorgehen ist recht kostengünstig und schnell zu bewerkstelligen. Dieser Trend lässt sich gut daran ablesen, dass heute alle großen ERP- und CRM-Hersteller schon SOAP-Interfaces, als Grundlage für Web-Services, für ihre Software unterstützen oder angekündigt haben. In diesem Sinne sind Webservices Geschäftsprozessimplementierungen, welche im Internet über wohldefinierte Interfaces als auch über standardisierte Internetprotokolle zur Verfügung stehen. Diese Webservices erlauben es einer Organisation, ihre Dienstleistungen einer großen Anzahl von Consumern zur Verfügung zu stellen. Durch die Nutzung von standardisierten Internetprotokollen geschieht dies in einer einfachen und effektiven Art und Weise. Im Vergleich zu den mehr allgemein orientierten Komponenten sind die Webservices ein spezieller Fall, denn hierunter versteht man lose gekoppelte, ausführbare Applikationen, welche dynamisch über ein TCP/IP-Protokoll eingebunden werden. Aus einer anderen Perspektive beurteilt, sind Webservices eine mögliche Implementierungsform von einer SOA. Die offizielle Definition von Webservices ist laut dem World Wide Web Consortium:

... software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet based protocols.

Die Nutzung von SOAP ist auch eines der Probleme der Webservices. Ursprünglich war SOAP entwickelt worden um Remote Procedure Calls (RPC) zu ermöglichen. Beim RPC wird ein lokaler direkter Aufruf eines Moduls durch einen Aufruf eines Moduls in einem anderen Rechner ersetzt. Diese Herkunft führte dazu, dass die ersten Webservices SOAP als „Tunnel“ für applikationsspezifische verteilte Aufrufe genutzt haben und damit keinen Servicecharakter zeigten. Diese Situation setzt sich heute fort, da Applikationen, die auf SOAP aufbauen, nicht per se interoperabel sind.

Webservices haben keine eigenen Benutzeroberflächen, mit der Folge, dass solche Benutzeroberflächen durch andere Mechanismen erst gebaut werden müssen. Daraus resultiert eine geringe Sichtbarkeit der Webservices für die Endbenutzer, da die Endbenutzer Software direkt nur durch die Benutzeroberflächen erleben. Welche Voraussetzungen an Technik auf der Protokollebene sind für die Webservices notwendig? Obwohl Webservices auf Standardprotokollen aufbauen, brauchen sie eine gewisse Menge von Voraussetzungen. Diese Voraussetzungen bauen systematisch aufeinander auf. Der sogenannte Web-Service-Stack erfüllt diese Voraussetzungen, er braucht, von unten nach oben betrachtet:

- TCP/IP – Diese logische Basisverbindung stellt das Rückgrat jeder Kommunikation im Webservice-Umfeld dar.
- XML – Die Protokollsprache XML dient zum Messageaustausch der einzelnen Webservice-Aufrufe.
- HTTP – Das HTTP nutzt das TCP/IP als darunterliegendes Transportprotokoll. Durch SOAP wird HTTP sowohl für den Aufruf sowie den Austausch der XML-Dateien beziehungsweise XML-Datenströme genutzt.
- SOAP
- UDDI.
- WSDL.

Aufbauend auf diesem Protokollstack werden die einzelnen Web-Services implementiert. Interessanterweise sind die Teile TCP/IP, HTTP, XML und SOAP so weit verbreitet, dass sie in vielen Bereichen den heutigen De-facto-Standard darstellen. Im Fall von TCP/IP ist der Übergang zur Hardware in Form von embedded Chips auf den Netzwerkkarten heute schon fließend. Obwohl eine Reihe von verschiedenen Internetprotokollen existiert, hat sich XML als der de facto Standard für Webservices herauskristallisiert. Historisch gesehen ist die Idee der Webservices nicht neu, schon CORBA hat ähnliche Mechanismen unterstützt. Was aber hierbei neu ist, ist die Einfachheit und Effektivität der Nutzung und der Gebrauch der standardisierten Internetprotokolle. Webservices basieren heute auf XML und dem plattformunabhängigen SOAP-Protokoll. Genauso wie HTML sind Webservices sehr einfach zu nutzen und hochflexibel. Sie sind einfach und gleichzeitig universell genug, damit sie von einer großen Anzahl von Consumern genutzt werden können. Diese Einfachheit hat sie so populär gemacht, dass sie schneller ein de facto Standard wurden, als es ihrem eigentlichen Reifegrad entspricht.

3.3 Prozess der Kundenanlage

Als Grundlage dieses Konzeptes dient der Prozess eines neuen Abonnenten, sprich eines neuen Kunden für den Profi-Fußballverein. Dieser Kunde erhält durch sein Abonnement den monatlichen Newsletter sowie die Vereinszeitschrift, kann im zugehörigen Onlineshop Merchandise-Artikel zu rabattierten Preisen erwerben und bekommt seinen eigenen Kundenbereich im Online-Portal des Vereins. Um diesen großen Prozess zu überschauen, ist es notwendig, diesen in weitere, kleinere, Teilprozesse zu zerlegen, um die Komplexität des Ganzen zu erfassen.

Der Gesamtprozess wird in drei Teile zerlegt:

- CRM-System
    - Abbildung der Kundenanlage im CRM-System,
    - Sugar-CRM als verwendete Software
    - Anlage der Stammdaten
    - Festlegen von weiteren Einstellungen für den Kunden, z.B. Empfang des Newsletter, Freischaltung für den Online-Shop, etc.
- CM-System
    - Abbildung des Kundenportals im CM-System
    - TYPO3 als verwendete Software
    - Login des Kunden, Prüfung gegen CRM-System
    - An-/Abmeldung Newsletter
- Onlineshop
    - Abbildung des Onlineshops
    - Magento als verwendete Software
    - Login des Kunden, Prüfung gegen CRM-System

Bei der Zerlegung ist nur auf einen reduzierten Satz an Prozessen eingegangen worden, da die Funktionalität des Systems jederzeit erweitert werden kann. Jeder der drei großen Sub-Prozesse beinhalten jeweils wieder mehr oder weniger kleinere Unterprozesse, siehe Abbildung 14. Diese können auch wieder Unterprozesse beinhalten, darauf wird allerdings in dieser Abhandlung aufgrund der Übersichtlichkeit verzichtet.

Abbildung 14: Prozesse für Kundenanlage

3.4 Die Komponenten

Da seitens des Kunden für die verwendete Software dem Realisierungspartner freie Hand gelassen worden ist, kommen nur Open-Source Softwarepakete zum Einsatz. Jedes einzelne Softwarepaket ist dabei mehrere Jahre am Markt, hat bewiesen, dass es stabil und verlässlich läuft und hat eine starke Community und ausreichend Partnerfirmen, um ggf. Support einzuholen. Grundsystem, an das alle anderen Systeme angebunden werden, ist der Enterprise Service Bus Mule. Dieser ESB, der als Middleware eingesetzt wird bildet das Kernstück und bietet für alle Systeme passende Web-Services an. Als weitere Softwarelösungen kommen das Content Management System TYPO3 sowie die eCommerce Plattform Magento zum Einsatz.

3.4.1 ESB – Enterprise Service Bus

Mule ist ein Java-basierter Enterprise Service Bus sowie eine Integrationsplattform, welche Entwicklern erlaubt, schnell und einfach Anwendung anzubinden, um Daten auszutauschen. Dabei verfolgt Mule den Ansatz einer Service-orientierten Architektur. Mule bietet eine einfache Integration von bestehenden Systemen, ganz egal welche Technologie(JMS, Web Services, JDBC, HTTP, usw.) das System nutzt.

Der ESB Mule wird in zwei verschiedenen Versionen angeboten: eine Community Edition und eine Enterprise Edition. Mule Community ist ein leichtgewichtiges Werkzeug, das alle Basisfunktionalitäten mitbringt, um externe Systeme zu integrieren. Mule Enterprise beinhaltet zusätzliche Features, welche sich besonders für Live-Umgebungen eignen, die besonderen Augenmerk auf Performanz, Skalierbarkeit, High Availability, Elastizität oder technischen Support legen.

Beide Versionen entspringen einer gemeinsamen Codebasis, sodass es problemlos möglich ist, von der Community Version auf die Enterprise Version upzugraden.

Mit dem ESB Mule versucht man die oft recht schwierige Arbeite der Integration maßgeblich zu vereinfachen. Ausgehend von der Architektur siehe Abbildung 15 kann Mule helfen die Komplexität der Ende-zu-Ende-Beziehungen aufzulösen.

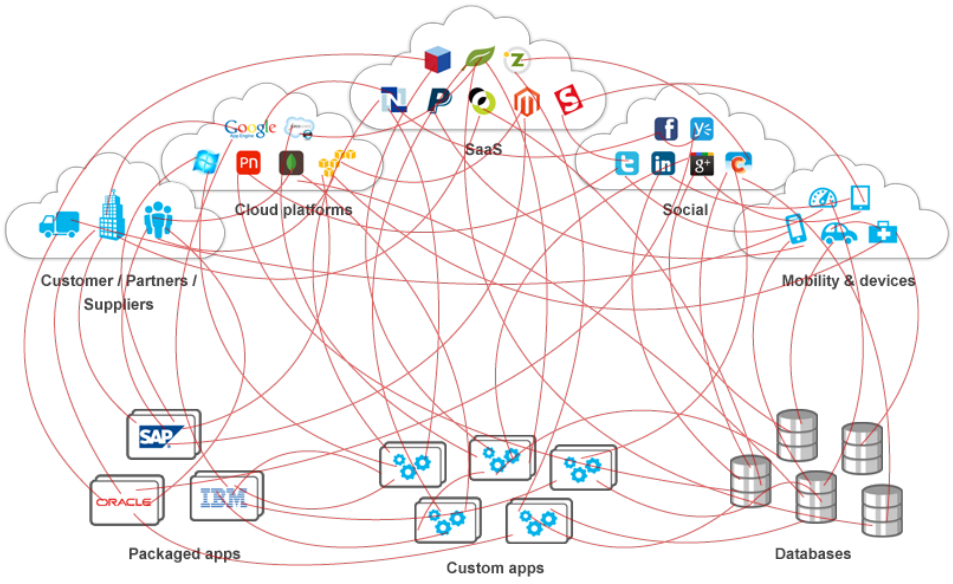

Abbildung 15: Architektur ohne ESB[12]

Bei der Einbindung des ESB Mule umgeht man den sehr hohen Aufwand jeden einzelnen Service mit jedem anderen zu verknüpfen. Es ist sogar unerheblich, ob die jeweilige Software inhouse gehostet wird oder in der Cloud. Mule bringt für nahezu

[12] (Meet Mule - Current Mule Documentation, 2013)

jeden Anwendungsfall sogenannte Connectors mit, die schon alle erdenklichen Grundverbindungen für die externen Web-Services mitbringen, siehe Abbildung 16.

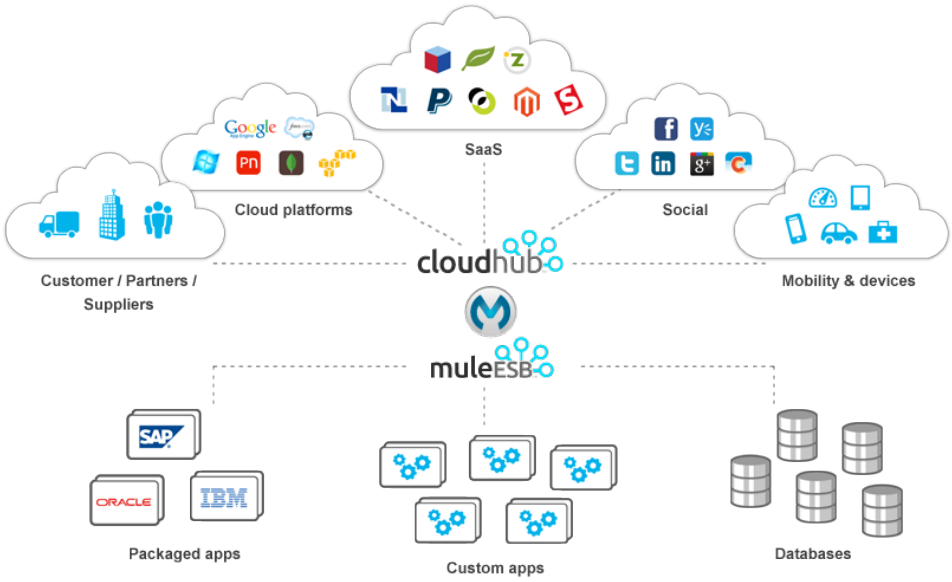

Abbildung 16: Architektur mit Mule und Connectoren[13]

Durch die Connectoren, die der ESB Mule für nahezu alle gängigen großen Software-Suiten bereitstellt, ist es problemlos möglich mit der API der jeweiligen Suite umzugehen. Ein Detailwissen über die API ist nicht mehr notwendig, da der Connector die Kommunikation nahezu automatisch durchführt. Grundangaben, die für jeden Kommunikationsversuch mit der API gemacht werden, sind:

- Authentifizierung mittels Benutzername und Passwort,
- Senden von Daten (optional),
- Empfangen von Daten in einem bestimmten Format (XML, JSON, etc.).

Mule ESB enthält von Haus aus eine Vielzahl an Konnektoren, die je ein Kommunikationsprotokoll unterstützen. Unter anderem werden folgende Protokolle unterstützt.

Mail	IP Transport	Datenbanken	Web	Web Services
IMAP	TCP	JDBC	HTTP(S)	WSDL, SOAP, REST, Axis
POP3	UDP	Hibernate	(S)FTP	.NET Web Services
SMTP	Multicast	LDAP	XMPP	CXF

[13] (Meet Mule - Current Mule Documentation, 2013)

Um Interaktionen mit Cloud- und Web-basierten Angeboten einfach realisieren zu können, bringt Mule ESB eine Reihe sogenannter Cloud Konnektoren mit. Unterstützt werden zum Beispiel die folgenden Systeme

- Amazon S3
- Facebook
- Flickr
- Paypal
- Salesforce
- Twitter
- SugarCRM
- Apple Push
- Bit.ly

sowie die eCommerce Plattform Magento.

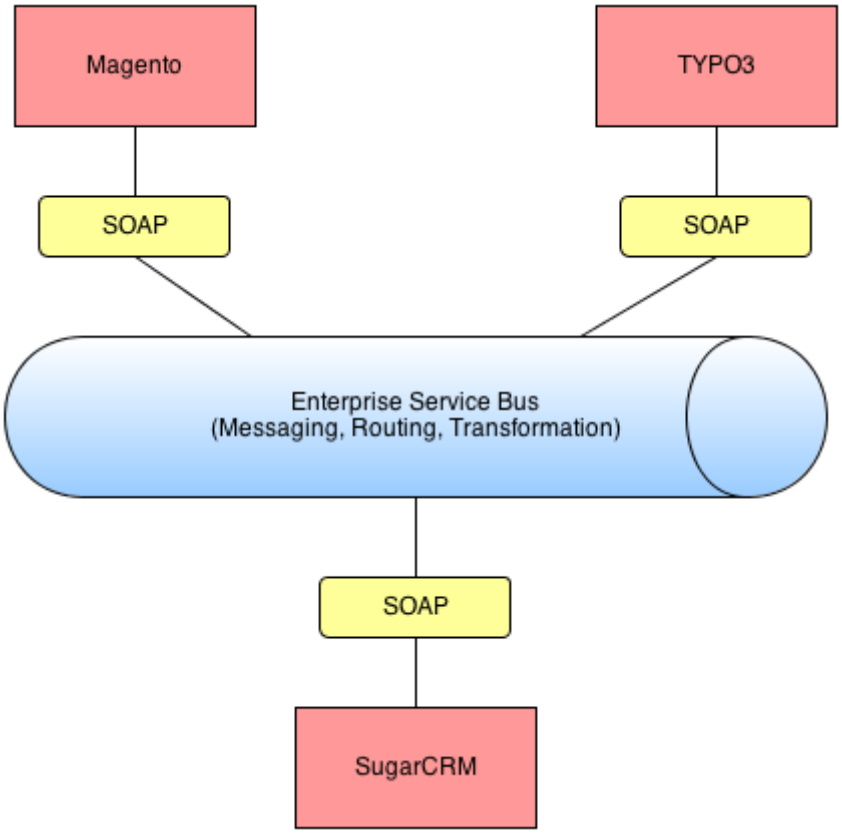

Abbildung 17: Anbindung der Systeme Magento, TYPO3 und SugarCRM an den ESB (eigene Darstellung)

Die Abbildung 17 zeigt einen schematischen Aufbau der insgesamt vier Systeme. Dabei wird angenommen, dass alle Systeme über SOAP/HTTP miteinander kommunizieren. Da es sich um Web-Services handelt, wäre es natürlich auch

möglich einzelne bzw. alle Systeme über REST kommunizieren zu lassen. Gänzlich unerheblich ist dabei auch der physische Standort der einzelnen Applikationen. Diese können inhouse zentral, inhouse dezentral oder auch in der Cloud gehostet werden.

4 Wirtschaftliche Aspekte

4.1 Vorteile einer SOA – Der Nutzen

Die Motivation eine Service Orientierte Architektur einzuführen ergibt sich aus vielen Vorteilen, die diese bieten. Eine SOA führt zu einem besseren Verständnis und einer höheren Transparenz der Geschäftsprozesse, die davon betroffen sind. Durch die Kapselung der Prozesse auf niedrigerer Ebene werden die an die Geschäftsprozesse orientierten IT-Systeme besser strukturiert. Des Weiteren ist durch die Vereinheitlichung eine gute Austauschbarkeit, Wartbarkeit und Erweiterbarkeit gegeben. Eine Qualitätsverbesserung durch bessere Testbarkeit und Vergleichbarkeit ist durch eine SOA außerdem auch gegeben. Ein Unternehmen kann durch eine SOA sehr schnell agieren und reagieren, da die Verantwortlichen schnell über Ereignisse, Prozesse und auftretende Fehler informiert werden. Ein weiterer wichtiger Vorteil ist, dass SOA Plattformunabhängig ist. So können unterschiedliche Technologien gekoppelt werden und plattformabhängige und herstellerabhängige Einschränkungen entfallen.

Zusammenfassend lassen sich folgende Vorteile herausheben:

- Kosteneinsparungen durch
  - Integration externer Services bei Business
  - Process Outsourcing
  - Outsourcing von exakt identifizierbaren Teilprozessen
  - Komplexitätsreduktion durch standardisierte Schnittstellen zwischen Systemen
  - einfachere Wartbarkeit der Systeme

- Einfachere Abbildung von
  - neuen Geschäftsprozessen
  - neuen Produkten

- Investitionsschutz durch
  - schnelle und kostengünstige Adaptierbarkeit von Legacy-Anwendungen
  - Erhöhung der Wartbarkeit

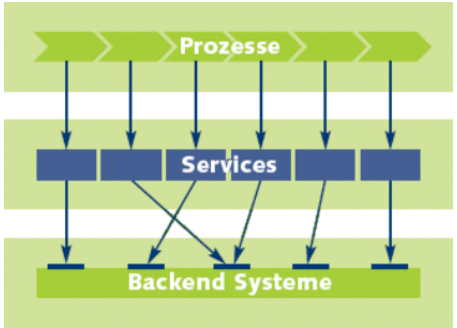

Abbildung 18: Vereinfachte Darstellung einer losen Kopplung von Prozessen, Services und Backend-Systemen

4.2 Allgemeine Herausforderungen - Umsetzung und Einführung einer SOA

Die Einführung einer SOA bietet viele Vorteile, jedoch ist eine SOA auch mit Herausforderungen und Nachteilen verbunden. Meist ist ein sehr hoher initialer Aufwand zu betreiben, wenn man die bestehende Architektur in eine SOA umbaut. Dabei kann es auch passieren, dass Einschränkungen bezüglich der Komponenten getroffen werden müssen, manche Komponenten müssen auch abgeändert werden. Hat man dann eine Architektur aus bestehenden Services, ist es schwer, ab einer gewissen Anzahl den Überblick zu behalten.

Dieses Problem lässt sich durch eine SOP beheben, doch ist auch die Integration einer solchen SOP meist nicht einfach. Ein wichtiger Punkt ist auch die Sicherheit, zwar haben sich einige Standards etabliert, jedoch ist es immer noch eine Herausforderung, Sicherheitsbestimmungen in SOA-Systeme zu integrieren. Auch die Verlässlichkeit von serviceorientierten Applikationen ist ein kritisches Thema. Die wohl wichtigste Eigenschaft einer SOA ist, dass ein Netzwerk, wobei meist das Internet gemeint ist, zum Austausch von Daten benötigt wird. Schlagworte wie Phishing, Spoofing, Viren oder Spam zeigen nur einige wenige Gefahren des Internets auf. Die Verlässlichkeit und Sicherheit von SOAs wird durch genau diese Gefahren stark beeinträchtigt. Die Kommunikation zwischen Services, besonders Webservices ist wegen dem Einsatz von Protokollen wie SOAP, HTTP und TCP relativ umfangreich in Bezug auf die Datenmengen.

Aus dem Blickwinkel der Enterprise Architektur lassen sich die Risiken für eine SOA in die einzelnen Teilarchitekturen einteilen:

- Geschäftsprozessarchitektur:
  Fehlerhafte SOA-Implementierung auf Grund anderer inflexibler Policies. Während der Aufbauphase haben Architekten oft das Problem, dass es Randbedingungen durch die Policies in der Organisation gibt, welche die Durchführung erschweren[14]. Typische Ursachen für diese Lücke sind:
  - Standards in der Organisation sind nicht nach dem Serviceparadigma spezifiziert.
  - Führungsebene verharrt in „altem“ Denken.
  - Die Konflikte zwischen einer SOA und den vorhandenen Policies werden weder eingeplant und noch aktiv kommuniziert.
  - Die Governance zeigt keine Wege zur Konfliktlösung auf.
  - Fehlende Überzeugungsarbeit bei den Betroffenen.

- Oft werden die QoS-Anforderungen einer einzelnen Applikation auf den gesamten Geschäftsprozess übertragen, mit der Folge, dass die gelieferte Qualität nicht mit der für jede Aktivität erwarteten übereinstimmt. Typische Ursachen sind: Mangelndes kognitives Alignment, keine Modellierung der

[14] So die Nutzung von Stored Procedures, welche einen Widerspruch zur Layerarchitektur darstellt.

geforderten Qualität aus Prozesssicht und keine Berücksichtigung der QoS-Aspekte.

- Informationsarchitektur: Das größte Problem auf der Ebene der Informationsarchitektur ist die fehlerhafte Abbildung zwischen dem Geschäftsobjektmodell und der Speicherung. Viele Softwareentwickler tendieren zu einer direkten Abbildung zwischen beiden Welten[15], mit der Folge von sehr ineffizienten Services. Umgekehrt führen Geschäftsobjekte, die aus dem Datenbankmodell abgeleitet wurden[16], zu einer Unmenge an schwer handhabbaren Services.
- Applikationsarchitektur:
    - Die fehlerhafte Identifikation der fachlichen Services stellt ein großes Problem dar. Oft werden Services aus IT-Sicht und nicht aus fachlicher Sicht identifiziert, was zu einer geringen Wiederverwendung und hohen Maintenance-Aufwänden führt. Ursache hierfür ist neben fehlendem Know-how ein mangelndes kognitives Alignment, da die SOA von der IT und nicht vom Fachbereich vorangetrieben wird.
    - Auch wenn die Services richtig identifiziert wurden, können fehlerhafte Implementierungen zu sehr hohen Kosten führen.
    - Ineffiziente Behandlung von Ausnahmen in den Prozessen. Jeder einzelne Geschäftsprozess kann Ausnahmen produzieren, diese müssen explizit modelliert und behandelt werden.
    - Oft wird von Werkzeugherstellern empfohlen, einfach einen Webservicewrapper einzusetzen und damit alle Legacysoftware in Services zu verwandeln. Ein solcher Schritt ist aus zwei Gründen falsch:
        - Jeder Bezug zum eigentlichen Geschäftsprozess oder zur Domäne geht faktisch verloren. Eine SOA basiert aber primär auf der Fachlichkeit, nicht auf der technischen Machbarkeit.
        - Die entstehenden Interfaces haben in der Regel reine CRUD-Funktionen.

      Ein ähnlich falscher Ansatz ist es, reine Datenservices auf der Datenbank zu schaffen und die Applikationen auf diese zugreifen zu lassen.
- Technologiearchitektur:
    - Die falsche Wahl eines ESBs ist ein großes Problem. Besonders durch die Softwarehersteller werden EAI-Systeme als ESBs verkauft und als zu erhaltende Investition müssen diese im Rahmen der SOA eingesetzt werden. Meist sind einfache Services noch gut handhabbar, aber Maintenance und Ausdehnbarkeit sind nur schwer zu gewährleisten.
    - Selbstgebaute Integrationsframeworks – In vielen Organisationen sind selbstentwickelte Frameworks vorzufinden, diese behindern eine SOA

[15] Diese direkte Abbildung wird von Entwicklungswerkzeugen sogar suggeriert.

[16] Der Begriff CRUD stammt aus der Datenmodellierung und steht für: Create, Read, Update, Delete. Dies sind die elementaren Funktionen bei dem Zugriff auf einem Datenelement oder Geschäftsobjekt.

mehr als das sie helfen, da sie genau wie einige kommerzielle ESBs eigentlich nicht als SOA ausgelegt sind.

- Fehlerhafte Deploymentstrategie – Oft wird eine explizite Deploymentstrategie nicht abgeleitet, eine solche ist aber für den langfristigen Einsatz unabdingbar.

In Bezug auf das oft gehörte Argument der möglichen Kostenersparung durch den Einsatz von SOA muss gesagt werden, dass dasselbe Argument bisher immer für die Einführung einer neuen Technologie herhalten musste und nie wirklich eintrat. Der Grad der möglichen Einsparung hängt primär vom Grad der Wiederverwendung von Services ab, ist also nicht durch die Architektur, sondern durch die Identifikation und Konstruktion der Services bedingt. Die schlechten Erfahrungen mit dem Maß an Wiederverwendung in der Objekt- und Komponentenorientierung lassen jedoch vermuten, dass auch bei der Serviceorientierung eine groß angelegte Wiederverwendung nicht wirklich einsetzen wird. Neben den Kosten sind die beiden Problemgebiete Performanz (so- wie ihre Prognose) und Sicherheit innerhalb einer SOA weder verankert noch in irgendeiner Weise adressiert. Der größte Fehler, der bei der Umwandlung eines bestehenden Systems in ein serviceorientiertes System gemacht werden kann, ist eine technikgetriebene Bottom-Up-Einführung, oft auch mit dem Euphemismus „organisches Wachstum“ belegt. Diese Strategie ist deswegen so beliebt, weil die Werkzeughersteller Generatoren für Webservices aus bestehendem Sourcecode anbieten und weil die IT den Fachbereich nicht involvieren muss. Typischerweise sind in Organisationen mit EAI-Systemen dann sogenannte Integrationsservices der Startpunkt für die Bottom-Up-Strategie. Diese Strategie hat aber massive Nachteile:

- Die entstehenden Services haben keinerlei organisationsweite Bedeutung. In den meisten Fällen sind sie nichts anderes als ein neues Interface für eine bestehende Applikation. Damit wird implizit eine enge Koppelung zwischen den bestehenden Legacysystemen und den Services erzeugt, welche eine Veränderung des Legacyportfolios deutlich schwieriger macht.
- Das bestehende Legacyportfolio hat in aller Regel ein hohes Maß an Redundanz mit der Folge, dass eine große Zahl an redundanten oder teilredundanten Services entsteht. Dasselbe Phänomen ließ sich in der Vergangenheit bei Datenbanken beobachten, auch dort ist ein Großteil der Attribute redundant.
- Die meisten Applikationen haben proprietäre Datenmodelle, welche durch Services organisationsweit abgebildet werden. Diese Proprietarität hat aber zur Folge, dass fast nur Punkt-zu-Punkt-Verbindungen entstehen.
- Die Implementierung eines Geschäftsprozesses nach Refactoring aus solchen Services ist oft unmöglich, da ein großer Teil des Wissens über den Prozess im Legacysystem (es ist schließlich ein soziotechnisches System) versteckt und verankert ist. Dies limitiert praktisch gesehen jede Form von Adaption.

Auch wenn Top-Down angefangen wird, früher oder später landen die Softwareentwickler bei dem Versuch, ein CRUD-Interface zu implementieren. Ein Grund hierfür ist, dass solche Operationen praktisch in jedem Design, von der

Structured Analysis über die Objektorientierung bis hin zur Komponentenbauweise auftauchen. Trotz der weiten Verbreitung bereiten die CRUD- Interfaces große Probleme:

- Das Interface neigt zur Verwendung in einem direkten Aufrufstil und nicht in einem Servicestil.
- Andere Services müssen dieses Interface öfters aufrufen, um tatsächliche Veränderungen zu produzieren.
- Es handelt sich um den falschen Abstraktionsgrad oder die falsche Granularität. CRUD-Operationen sollten innerhalb eines Services faktisch privat geschehen.
- Das Read-Interface führt bei großen Datenmengen entweder zu Problemen oder zum Versuch der Implementierung eines Pointers, der durch den Aufrufer gesteuert werden kann.
- Die zugrundeliegenden Daten können sehr leicht inkonsistent werden, da das CRUD-Interface nichts über den Kontext der Operation und die daraus notwendigen Randbedingungen weiß.

5 Schlusswort

Ziel der Arbeit war es, ein Verständnis für die service-orientierte Architektur, SOA, zu schaffen und basierend auf dieser ein Konzept für die IT-Struktur eines Profi-Fußballvereins zu schaffen. Dabei ist sehr detailliert auf die SOA eingegangen worden, da es, trotz zahlreicher Literatur und Vorträgen, recht schwierig ist, genau zu verstehen was SOA ausmacht und warum SOA abgegrenzt werden muss, zu anderen Konzepten, die in der Informatik und deren Architekurtypen bestehen.

Einmal konzipiert und eingeführt, bietet die SOA, mit den Systemen die in Kapitel 3 beschrieben werden, eine sehr solide und belastbare Basis, die auch zukünftigen Anforderungen an das Softwaredesign und speziell deren gegenseitige Anbindung gerecht wird. Die Nutzung des Enterprise Service Bus Mule als Middleware ist dabei von entscheidendem Erfolg für die komplette Architektur. Ohne diesen Kern, welcher an sich schon nach SOA-Konzepten entworfen ist, wäre es nicht möglich, in relativ kurzer Zeit, ein flexibles und in den Möglichkeiten der Anbindung so vielfältiges System zu errichten.

Quellenverzeichnis

Meet Mule - Current Mule Documentation. (02. 09 2013). Von http://www.mulesoft.org/documentation/display/current/Meet+Mule abgerufen

BECKER, A. (2010). *Nutzenpotenziale und Herausforderungen Service-orientierter Architekturen.* Wiesbaden: Gabler Verlag.

BISKE, T. (2008). *SOA Governance.* Birmingham: Packt Publishing Ltd.

JACOBS, I., & WALSH, N. (15. Dezember 2004). Architecture of the World Wide Web, Volume One, W3C Recommendation.

LIEBHART, D. (2007). *SOA goes real.* München Wien: Carl Hanser Verlag.

MASAK, D. (2007). *SOA?* Berlin Heidelberg: Springer-Verlag.

MELZER, I. (2010). *Service-orientierte Architekturen.* Heidelberg: Spektrum Akademischer Verlag.

MONROE, R. T., KOMPANEK, A., MELTON, R., & GARLAN, D. (1997). *Architectural Styles, Design Patterns and Objects.* IEEE Software.

PERRY, D. E., & WOLF, A. L. (1992). *Foundation for the Study of Software Architecture.* ACM SIGSOFT Software Engineering Notes Vol 17 Issue 4.

SCHMALE, T. (2004). Mit EAI und SOA zum Real-Time Enterprise, Buchbeitrag zu "EAI - Serviceorientierung und nachhaltige Architektur. inubit AG.

SHAW, M., & GARLAN, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline.* USA: Prentice Hall.

Ehrenwörtliche Erklärung

„Ich erkläre hiermit ehrenwörtlich",

1. dass ich meine Bachelorthesis mit dem Thema

   „Konzipierung einer serviceorientierten Software-Architektur mit Hilfe einer ESB-Middleware für einen Profi-Fußballverein“

   ohne fremde Hilfe angefertigt habe,

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und

3. dass ich meine Bachelorthesis bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Chemnitz, 16.09.2013

Ort, Datum

Unterschrift