1 Grundlegung

1.1 Grundlegende Definitionen

Kosten und Komplexität in der IT wachsen - doch eine gründliche Analyse und Planung wird oft sträflich vernachlässigt. Dies führt dazu, dass laut einer repräsentativen Studie aus dem Jahr 2004 der Standish Group in der Regel 72% der IT-Projekte scheitern oder die Kosten und die Zeit aus dem Ruder laufen (Standish Group 2004). Somit werden IT-Architekturen in Unternehmen, gerade für eine erfolgreiche IT-Governance, umso wichtiger.

Es gibt ein großes und oft verwirrendes Angebot an Methoden zum Erstellen und Überwachen von IT-Architekturen, doch meist sind die grundlegendsten Begriffe unklar oder unbekannt. Dies trifft nicht nur auf IT-Experten zu, sondern vielmehr noch auf IT- und Fachabteilungen, die sich nicht verständigen können und so aneinander vorbei arbeiten.

Architektur stammt aus dem Griechischen und setzt sich zusammen aus:

- *arché* - Ursprung, Anfang
- *techné* - Handwerk, Kunst, Fertigkeit

Das bedeutet, dass Architektur als „Erste Kunst“ oder „Oberste Fertigkeit“ angesehen werden kann und ein Architekt (architékos) Handwerker und Künstler zugleich sein muss, um komplexe Strukturen zu erschaffen. Dies wiederum erfordert umfangreiches Wissen darüber, was erschaffen wird und darüber, wie es zu erschaffen ist. Er ist das Bindeglied zwischen dem Nutzer und dem Handwerker.

Für die Informationstechnologie bedeutet Architektur, ein Konzept zu besitzen, das eine zielorientierte Entwicklung, Wartung, Dokumentation und Nutzung des IT-Systems ermöglicht. Warum sind 72% der IT-Projekte Misserfolge, wenn man doch auf die IT-Architektur verzichten kann? Es ist an der Zeit, das Ruder herumzureißen und vielmehr Architekten in die Planung der IT-Systeme einzubinden, um in Zukunft mindestens 72% Projekterfolge zu verzeichnen.

Der Architekturbegriff wird in der einschlägigen Literatur bislang ausgesprochen uneinheitlich verwendet. Der Versuch, eine brauchbare Definition aus der Literatur zu (re-)konstruieren, scheitert daher. Ein Blick über den informatischen Tellerrand hilft jedoch weiter. Stellt man sich die Frage, was ein Architekt zuerst vorlegt, unabhängig davon, ob er eine Hundehütte oder einen Wolkenkratzer baut, dann ist dies ein Modell, das *ganzheitlich* zeigt, wie das komplexe Gesamtwerk nach seiner Vollendung aussehen wird. Als Architektur wird daher ein ganzheitliches Modell eines komplexen Informatik-Systems verstanden. Ein *Modell* ist ein abstraktes immaterielles Abbild realer Strukturen bzw. des realen Verhaltens für Zwecke des Subjekts. Modelle dienen damit dem Erreichen von Zielen (Zweckorientierung) und sind subjektiv auf den Modelladressaten zugeschnitten. Der Modelladressat von Architekturen ist der Systemgestalter, dessen Ziel es ist, neue In-

formatik-Systeme nutzbringend und im vorgesteckten Projektrahmen erfolgreich zu realisieren. Als Informatik-System wird hier jede Art von Technologie verstanden, die von Informatikern gestaltet werden. Die Palette reicht hier von Informationssystemen, die direkt der Leistungserstellung durch den Nutzer dienen, über Basissoftware, die Grundlage für Entwicklung und Betrieb von Anwendungssystemen ist (z. B. Datenbanksysteme, Utilities, Betriebssysteme usw.), bis zu Hardwaresystemen. Informationssysteme, Basissoftware und Hardware bilden zusammen die technische *Informationsinfrastruktur*.

Entsprechend der oben angegebenen Klassifikation von Informatik-Systemen lassen sich auch die zugrunde liegenden Architekturen aufschlüsseln:

- *Informationsarchitekturen* stellen die Beziehungen zwischen Teilen eines Informationssystems oder verschiedener Informationssysteme dar.
- *Softwarearchitekturen* zeigen die Zusammenhänge zwischen Elementen von Softwaresystemen.
- *Rechnerarchitekturen* beschreiben in Analogie zu den Softwarearchitekturen die Zusammenhänge zwischen Hardwarekomponenten.

Alle drei Architekturen bilden zusammen einen „Masterplan“ für die Gestaltung einer Systemlandschaft, der hier als *IT-Systemarchitektur* bezeichnet wird. Sie ist die Grundlage für die Entwicklung, Pflege und Wartung der Informationsinfrastruktur einer Organisation und damit der Schlüssel für ein erfolgreiches Informationsmanagement.

1.2 Konventionelle und dienstorientierte Informationssysteme

Die Begriffsdefinitionen aus Abschnitt 1.1 Grundlegende Definitionen haben noch einige Schönheitsfehler, da recht breit interpretierbare – man könnte auch „schwammige“ sagen – Termini wie „Teile eines Informationssystems“ oder „Elemente von Softwaresystemen“ verwendet werden. Der Grund hierfür ist, dass in diesem Werk von der konventionellen Sicht auf Informationssysteme abgerückt wird.

Als *konventionelle Informationssysteme* werden hier Softwaresysteme bezeichnet, die als ganze zusammenhängende Softwareartefakte konstruiert, ausgeliefert und installiert werden. Beispiele hierfür sind Office-Pakete oder ERP-Systeme (ERP = Enterprise Resource Planning). Konventionelle Informationssysteme sind für Anbieter wie auch Nutzer bequem. Sie werden als Boxware mit CDs oder DVDs und Handbüchern ausgeliefert, der Kunde zahlt eine Lizenzgebühr hierfür und hat „seine“ Software auf „seiner“ Hardware installiert. Dieses Geschäftsmodell führt jedoch zu einer gewaltigen Ressourcenverschwendung, da der Nutzer in der Regel nur einen Bruchteil der bereitgestellten Funktionalität nutzt und dies oftmals auch nur über eingeschränkte Zeiträume.

Beispiele hierfür sind Textverarbeitungsprogramme wie Microsoft Word®, von denen selbst erfahrene Buchautoren nur bis zu 70% der Funktionalität nutzen oder ERP-Systeme wie SAP R/3, wo zunächst mit aufwändigen Customizing-Prozeduren beträchtliche Anteile der Funktionalität ausgeklammert werden und von der

verbleibenden Funktionalität trotzdem nur ein Bruchteil verwendet wird. Beispiel für eine zeitweise Nutzung von Software ist die Funktion „Jahresabschluss“, die naturgemäß nur einmal pro Jahr verwendet wird.

Heute hat jeder Lizenznehmer seine Software bei sich (oder seinem Outsourcing-Dienstleister, was das Problem aber nur verlagert) unabhängig von der Nutzungsdurchdringung (prozentualer Anteil genutzter Funktionen zur Gesamtfunktionalität) oder Nutzungsintensität (Häufigkeit der Nutzung von Funktionen pro Zeiteinheit) vollständig installiert. Ungenutzte Funktionen verbrauchen nicht nur Ressourcen wie Speicherplatz, sondern verursachen auch Betriebskosten, etwa bei Release-Wechseln.

Die Alternative zu diesem Szenario sind *dienstorientierte Informationssysteme*. Aus Nutzersicht erscheinen sie folgendermaßen:

Der Benutzer meldet sich an einem Portal im Internet an, das Dienste anbietet, die für seine momentane Lebenslage relevant sind. Eine Lebenslage könnte sein, dass er gerade an seinem Arbeitsplatz sitzt und einen Kundenauftrag anlegen soll, er zuhause an seiner Steuererklärung arbeitet oder ein geeignetes Geschenk für seinen Sohn braucht. Er weist sich gegenüber dem Portal mit einer Chipkarte aus, die Informationen über seine momentane Lebenslage enthält (für den ersten Fall bedeutet dies z. B. dass er Mitarbeiter der TopMe GmbH ist, dort als Kundensachbearbeiter tätig ist und ihm das Profil „Sales“ im ERP-System zugeordnet ist). Auf der Grundlage der Lebenslage werden von der Portalsoftware alle Dienste, die der Nutzer braucht und auch benutzen darf, sowohl von lokalen Systemen als auch aus dem Internet zusammengesucht und an der Benutzeroberfläche zur Verfügung gestellt. Der Benutzer nutzt die Funktionen so, wie es im Workflow vorgesehen ist, und bemerkt keinen Unterschied zwischen lokalen Funktionen und solchen, die aus dem Internet hinzugefügt werden.

Folgendes unterscheidet ihre Nutzung von der konventioneller Informationssysteme:

1. Der Nutzer bekommt nur und genau das angeboten, was er in seiner Lebenslage braucht.
2. Der Nutzer authentifiziert sich nur durch seine Lebenslage. Da weder er selbst noch das Portal zum Zeitpunkt der Authentifizierung wissen, welche Dienste die Anfrage aus der Lebenslage befriedigen, kann sich der Nutzer auch nicht gegenüber den einzelnen Diensten authentifizieren. Dies wäre in der Praxis auch zu aufwändig.

Aus Sicht eines Systembetreibers sind die Unterschiede gravierender. Da die Trennung zwischen lokal installierten Funktionen und Diensten, die von Informationssystemen im Internet bereitgestellt werden, in dienstorientierten Informationssystemen aufgehoben ist, reicht es aus, nur noch Funktionen mit hoher Nutzungsdurchdringung und -intensität lokal vorzuhalten. Weiterhin sollten Funktionen, die Unternehmen Wettbewerbsvorteile sichern, lokal gehalten werden und nach außen abgeschirmt werden. Den „Rest“ lädt man sich über das Internet herunter, wenn man ihn braucht.

Die Konsequenzen einer solchen Softwarelandschaft sind dramatischer, als sie auf den ersten Blick erscheinen. So braucht heute beispielsweise jeder Nutzer ei-

nes ERP-Systems bei Nutzung konventioneller Informationssysteme eine vollständige Kopie des Systems, die letztendlich alle denkbaren Funktionen der betrieblichen Datenverarbeitung anbietet. n-Tausend Kunden von ERP-Software betreiben damit mindestens n-Tausend Kopien von dieser. Geht man davon aus, dass allein der ausführbare Kode eines ERP-Systems heute bis zu 1 GB groß sein kann, werden gegenüber dienstorientierten Informationssystemen n TB an Speicherplatz verschwendet. Das erscheint an sich nicht so dramatisch, kostet doch Speicherplatz praktisch „nichts“. Die Kostenwirkung wird aber deutlich, wenn man sich überlegt, dass n TB Anwendungskode nicht mehr customized, administriert und gewartet werden müssen.

An die Stelle der Lizenzierung ganzer Informationssysteme tritt die (zeitweise) Abonnierung von einzelnen Diensten. Damit lizenziert der Kunde nur noch die Funktionalität, die er wirklich braucht und das ggf. nur für einen bestimmten Nutzungszeitraum. Weiterhin hat der Kunde die Wahl, ob er einen Dienst selbst betreibt oder vom Anbieter oder einem Dritten betreiben lässt. Dieses Geschäftsmodell schafft Flexibilität und reduziert die Abhängigkeit von Kunden gegenüber einem oder wenigen Anbietern.

Das klassische Anwendungssystem mit einer fest umrissenen Funktionalität für einen bestimmten Aufgabenbereich hat damit ausgedient. Spätestens seit Erscheinen des Unternehmensdatenmodells (Scheer 1988) ist klar, dass es in Unternehmen keine Bereiche gibt, die nicht informatorisch mit (allen) anderen Bereichen verbunden sind. Nach der Automation betrieblicher Funktionen wurde daher die Integration von betrieblichen Anwendungssystemen zum führenden Paradigma der betrieblichen Informationsverarbeitung. Unternehmensdatenmodelle erwiesen sich allerdings aufgrund ihrer enormen Komplexität und des damit verbundenen Erstellungsaufwands als ungeeignet für die Integration der Informationsverarbeitung (Mertes/Klonki 1991). Auch die Punkt-zu-Punkt-Integration zwischen verschiedenen Anwendungssystemen erwies sich als untauglich. Die Vielzahl von Schnittstellen führte zur „Spaghetti-Integration“ mit allen aus der Spaghetti-Programmierung bekannten Wartungsproblemen – nun aber auf die interoperable Ebene transponiert.

An die Stelle von Unternehmensdatenmodellen und Punkt-zu-Punkt-Integration rückte das Paradigma der (Geschäfts-)*Prozessorientierung*, das heute erfolgreich bei der Entwicklung und Einführung von (Standard-)Software eingesetzt wird. Die Funktionalität von betrieblichen Anwendungssystemen orientiert sich jedoch traditionell an aufbauorganisatorischen Einheiten, z. B. Finanzbuchhaltung, Vertrieb, Produktion usw., was angesichts der Tatsache, dass Geschäftsprozesse kaum Rücksicht auf aufbauorganisatorische Gegebenheiten nehmen, nicht mehr zeitgemäß ist. So wirkt die aufbauorganisatorische Zuordnung von Funktionen auf Anwendungssysteme heute künstlich und erschwert die prozessorientierte (Re-)Organisation von Unternehmen. Mit der Prozessorientierung verschwimmen die Grenzen zwischen betrieblichen Anwendungssystemen.

Weiterhin macht die Integration nicht mehr an der Unternehmensgrenze halt. Wertschöpfung geschieht in Unternehmensverbunden von Lieferanten, Produktionsunternehmen, Handelsunternehmen und Dienstleistern. Auch der Endkunde ist in dieses Netzwerk mit einzubeziehen. Das Electronic Business (E-Business) bildet einen Ordnungsrahmen für die zwischenbetriebliche Integration von Ge-

schäftsprozessen. Systeme des Supply Chain Managements (SCM) oder des Customer Relationship Managements (CRM) sind Beispiele für Informationssysteme, die über den Rand der betrieblichen Informationsverarbeitung herausragen. Das E-Business ist daher ein geeignetes Anwendungsfeld für die Gestaltung dienstorientierter Informationssysteme. Werden Informationssysteme aus Diensten zusammengefügt, spielt es keine Rolle mehr, ob Dienste für die inner- oder zwischenbetriebliche Integration herangezogen werden.

Auch aus Anbietersicht sind dienstorientierte Informationssysteme attraktiv. So braucht ein Erfinder eines neuen computergestützen Verfahrens nicht mehr das gesamte Informationssystem um sein Verfahren herum zu entwickeln, um dieses zu vermarkten. Stattdessen kann er das Verfahren als Dienst bereitstellen und vermarkten, so dass seine Markteintrittsbarriere erheblich niedriger ist als bei konventionellen Informationssystemen. Dies kommt vor allem kleinen und mittleren Softwareunternehmen zugute.

1.3 Technologien für dienstorientierte Informationssysteme

Die Idee, Dienste aus der (Internet-)Steckdose anzubieten und für jeden nutzbar zu machen, ist nicht neu. Schon auf der ersten und legendären Software-Engineering-Konferenz 1968 in Garmisch-Partenkirchen wurde die Idee präsentiert, Software in Zukunft nicht mehr durch vollständige Neuentwicklungen, sondern durch das Zusammenstecken von standardisierten Softwarekomponenten zu erstellen. Die Idee der Komponente mündete zunächst im objektorientierten Paradigma, in dem Objekte als Kapselungen von Datenstrukturen und Funktionen (dort Methoden genannt) verstanden werden, die untereinander über Nachrichten miteinander kommunizieren. Werden Objekte über verschiedene Netzknoten verteilt und über eine gemeinsame Middleware (Object Request Broker – ORB) bereitgestellt, spricht man von verteilten Objekten. Der ORB erlaubt dann die Nutzung von Objekten „aus der Steckdose“ (Grotehen/Dittrich 1995).

Objekte sind allerdings nur in ihren objektorientierten Laufzeitumgebungen lebensfähig und haben damit keine hinreichenden Eigenschaften für die allgemeine Realisierung des Komponentenparadigmas. Aus Sicht dieses Paradigmas ist es unerheblich, auf welcher softwaretechnischen Grundlage eine Komponente implementiert wurde. Es muss lediglich sichergestellt werden, dass alle Leistungen, die eine Komponente für ihre Umgebung erbringt, über standardisierte Dienste zur Verfügung gestellt werden. Die Trennung zwischen Leistungserbringer und Leistungserbringung ermöglicht die Realisierung von Softwarearchitekturen auf Basis von Diensten, auch *Service-oriented Architecture* (SoA) genannt.

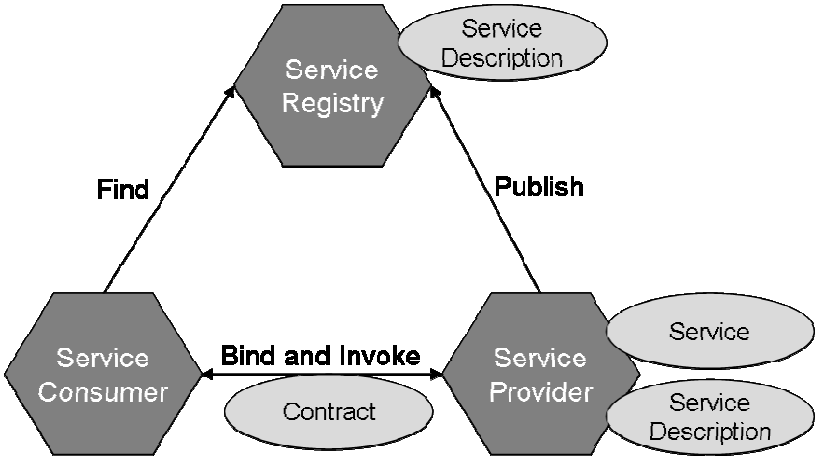

Abbildung 1: Grundmodell der SoA

Abbildung 1 zeigt das Grundmodell der SoA nach (McGovern et al. 2003). Es besteht aus den folgenden Elementen, die in einer SOA miteinander zusammenwirken (Budde 2005):

- *Service-Anbieter*: Anbieter eines bestimmten und im Netzwerk adressierbaren Dienstes. Er führt die Anfragen des Service-Konsumenten aus (*invoke* Operation) und veröffentlicht seinen Service beim Service-Verzeichnis (*publish* Operation).
- *Service-Konsument*: Dienstnachfrager, der den Service-Anbieter mittels einer Serviceanfrage (*invoke* Operation) aufruft. Dies kann beispielsweise ein Anwendungssystem oder ein anderer Service sein. Der Service-Konsument findet einen Service durch eine Anfrage an das Verzeichnis über die Operation *find*. Im Anschluss findet der Aufruf dieses Services beim Service-Anbieter statt; die Festlegung auf das zu nutzende Transportprotokoll erfolgt durch den *Binding*-Schritt.
- *Service-Verzeichnis*: Repräsentiert einen speziellen Dienst, auch als Maklerdienst bezeichnet, der das Nachschlagen eines Service ermöglicht (*find*). Es beinhaltet ein Verzeichnis aller veröffentlichten Services und ermöglicht die Suche innerhalb des Verzeichnisses.
- *Contract*: Ein Contract (Vertrag) ist eine Spezifikation der Kommunikation zwischen Service-Konsument und Service-Anbieter. Er legt das Format der Anfrage und der Antwort fest. Zusätzlich können Vor- und Nachbedingungen an dieser Stelle definiert werden, die vor dem Aufruf des Services gelten sollen. Des Weiteren können hier auch Angaben über die Dienstgüte (Quality of Service - QoS) hinterlegt werden.

Es ist allerdings eine Illusion, dass Software ausschließlich durch das Aufrufen von kleinen, überschaubaren und gut wartbaren Diensten entstehen kann. Die Komplexität verlagert sich in diesem Fall von der Implementierung zu den Schnittstellen zwischen den Diensten. Dienstorientierte Informationssysteme brau-

chen daher leistungsfähige *Mediatoren*, die das Zusammenspiel (auch Orchestrierung genannt) von Diensten koordinieren.

1.4 Ziel und Aufbau des Buchs

Damit ist das technische Szenario für dieses Werk umrissen. Ziel dieses Buchs ist es, eine theoretisch fundiert hergeleitete Methodik zur Konstruktion dienstorientierter Softwarearchitekturen zu präsentieren und anhand realer Beispiele zu illustrieren und zu validieren. Das Anwendungsszenario E-Business wurde gewählt, da hier ein besonders hoher Bedarf an interorganisationeller Integration der Informationsverarbeitung gegeben ist und die beteiligten Informationssysteme hinreichend komplex und kompliziert sind.

Im zweiten Kapitel werden die wesentlichen theoretischen Grundlagen für die Informationsverarbeitung im E-Business eingeführt. Hierzu gehören die wesentlichen Begriffsdefinitionen zu Information und Integration, die Konstruktion von Lebenslagenmodellen und die Grundlagen komponentenorientierter Softwarearchitekturen. Den Abschluss des Kapitels bildet eine Abhandlung zur Definition und Einordnung des E-Business.

Kapitel 3 behandelt die technischen Grundlagen. Einer kurzen Einführung in die Modellierungssprache UML, mit der alle nachfolgenden Konzepte der Softwarearchitekturen beschrieben werden, folgen

- Beschreibungen von *Mustern* (Pattern), die der formalisierten Spezifikation von Komponenten dienen,
- *Web-Services*, die Softwarekomponenten im WWW implementieren,
- *Agententechnologien*, die für die Kommunikation zwischen den verschiedenen Diensten eingesetzt werden, und
- *Peer-to-Peer (P2P)-Technologie*, auf deren Grundlage verteilte dienstorientierte Informationssysteme aufgebaut werden.

In Kapitel 4 wird dann zunächst detailliert ein Mediator spezifiziert, der Kernkomponente für die Realisierung dienstorientierter Informationssysteme ist. Danach wird schrittweise gezeigt, wie mithilfe der in Kapitel 3 vorgestellten Technologien verteilte dienstorientierte E-Business-Informationssysteme aufgebaut werden.

Kapitel 5 zeigt dann Realisierungen derartiger Informationssysteme. Zunächst wird IDSS (Intelligent Driver Support System) vorgestellt. Dies ist ein Fahrerassistenzsystem im mobilen E-Business (M-Business), in dem ein Mediator Dienste aus dem Internet, aus einem Navigationssystem und dem Bordcomputer eines Automobils integriert. Das zweite Beispiel ist SHERPA (SHared ERP Architecture), das eine ERP-Implementierung als dienstorientiertes Informationssystem darstellt.

http://www.springer.com/978-3-540-25821-6

Software-Architekturen für das E-Business
Enterprise-Application-Integration mit verteilten Systemen
Herden, S.; Marx Gómez, J.; Rautenstrauch, C.; Zwanziger, A.
2006, X, 192 S. 60 Abb., Softcover
ISBN: 978-3-540-25821-6