Technische Universität Graz (TUG)
Fakultät für Informatik
Lehrstuhl Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Wotawa

Umsetzung serviceorientierter Architekur für Anlagenmonitoring

Diplomarbeit
von

Florian Jöbstl

01. Februar 2010 – 20. Oktober 2010

Betreuer: Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Wotawa
Kooperationspartner: NTE Naturenergie. Technology and Engineering GmbH

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

……. ……..
date (signature)

Inhaltsverzeichnis

Zusammenfassung

Die Anforderungen an Softwareprojekte sind in Vergangenheit stetig gestiegen und werden es auch in Zukunft weiter tun. Um diese Komplexität überblickbar zu halten haben sich Programmiersprachen, Paradigmen und Netzwerktechniken im Laufe der Zeit evolutionär entwickelt. Der Ansatz der serviceorientierten Architektur verspricht Komplexität überschaubar zu halten und flexibel genug zu sein auf unweigerliche Änderungen reagieren zu können.

Kapitel 1 erläutert die Entwicklung von Softwarearchitekturen sowie die Motivation hinter SOA und geht dann auf die Grundbausteine einer SOA ein. Nach dieser Definition wird eine Kategorisierung von Diensten vorgestellt, die zu verschiedenen Ausbaustufen dieser Architektur führt. Abschießend wird auf die Qualitätsattribute, die diese Architektur interressant machen, eingegangen.

Kapitel 2 stellt Webservices als Technologie zur Umsetzung von SOA vor. Beleuchtet werden hier SOAP, WSDL, UDDI als Basis von Webservices und die Webservicestandards WS-*, die stark zu Interoperabilität von Webservices beigetragen haben.

Kapitel 3 befasst sich mit der Umsetzung von SOA für NTE.CLM, einer Plattform zur Überwachung von Energiedaten unterschiedlicher Anlagen. Dazu werden Anforderungen an die Architektur aus den Systemrollen erhoben und den Qualitätsattributen gegenübergestellt. Dienste, die im Rahmen dieser Applikation entwicklet wurden, und ihre Einteilung in die zuvor vorgestellten Kategorien werden beschrieben.

Abschießend zieht Kapitel 4 Bilanz über Erwartungen und Umsetzung von SOA sowie das Weiterentwicklungspotenzial der CLM-Plattform in der Zukunft.

Abstract

The requirements for software projects have constantly risen and will do so in future, too. In order to keep an overview of this very tendency, programming languages, paradigms and network techniques have developed evolutionary. The approach to use service-orientated architecture promises to keep complexities manageable and to be flexible enough to react to inevitable changes.

Chapter 1 will examine the development of software architectures, the motivation behind SOA and will then explain the basic modules of a SOA. After this definition, a categorization of services that leads to different stages of this architecture will be introduced. Finally, the quality characteristics which make this architecture interesting will be discussed.

Chapter 2 will introduce Web Services as a technology for implementation of SOA. SOAP, WSDL and UDDI will be discussed as bases of Web Services and their standards WS-*, as they have contributed strongly to the interoperability of web-services.

Chapter 3 will deal with the implementation of SOA for NTE.CLM, which is a platform for supervision of energy data from diverse facilities. In addition, requirements from the architecture are collected from the system roles and compared to the quality characteristics. Finally, services which have been developed in the frame of this application and their division into the categories formerly introduced will be described.

To conclude, chapter 4 will draw a balance about expectations and implementation of SOA as well as the potential of further development of the CLM-Platform in the near future.

Kapitel 1

Serviceorientierte Architektur

1.1 SOA als abstraktes Konzept

Die Anforderungen an Softwareprojekte sind in Vergangenheit stetig gestiegen und werden es auch in Zukunft weiter tun. Um diese Komplexität überblickbar zu halten haben sich Programmiersprachen, Paradigmen und Netzwerktechniken im Laufe der Zeit evolutionär entwickelt. Als Assembler für die Abbildung von Systemanforderungen an den Grenzen angelangte, war die prozedurale Programmierung das ablösende Paradigma. Es erlaubte Probleme in einfachere Teilprobleme zu zerlegen. Diese Abstraktion, sowie die Möglichkeit der Verteilung von Code in mehrere Files, führte erstmals zu wiederverwendbaren Algorithmen und so zu Bibliotheken.

Dijkstra beschreibt den damals vorherrschenden Zeitgeist in der Programmierung in „Notes on structured programming“ wie folgt:

> „Programming a large system in any typical programming language available today is an exercise in obscuration. We work hard at discovering inherent structure in a problem and then structuring our solution in a compatible way.“ [Dij72]

Diese Vorgehensweise führte zu dieser Zeit noch zu adäquater Software. Doch der Bedarf nach mehr Abstraktion formulierten z.B. DeRemer und Kron in „Programming-in-the-large versus Programming-in-the-small“:

> „Current languages discourage the accurate recording of the overall solution structure; they force us to write programs in which we are so preoccupied with the trees that we lose sight of the forest, as do the readers of our programs!“ [DK75]

In der objektorientierten Programmierung kapselte man Funktion und Daten in Objekten und beschrieb ihre Interaktion. Doch auch die Objektsicht auf ein System ist in vielen Fällen zu fein-granular und führt zu Komponenten, d.h. Subsystemen, die über eine wohldefinierte Schnittstelle interagieren.

Eine ähnliche Entwicklung durchliefen Netzwerktechnicken. Bereits 1976 formulierte James E. White im Request for Comments den Grundgedanken des entfernten Funktionsaufrufes:

> „The procedure call model would elevate the task of creating applications protocols to that of defining procedures and their calling sequences. It would also provide the foundation for a true distributed programming system (DPS) that encourages and facilitates the work of the applications programmer by gracefully extending the local programming environment, via the RTE, to embrace modules on other machines."This integration of local and network programming environments can even be carried as far as modifying compilers to provide minor variants of their normal procedure-calling constructs for addressing remote procedures." [Res76]

Diese Technik koppelte jedoch Aufrufer und Aufrufenden zu stark miteinander. Auch sind die verschiedenen Implementierungen wie z.B. Java RMI, COM/DCOM und .NET Remoting untereinander inkompatibel. Eine sprachen-unabhängige Weiterentwicklung dieses Konzepts stellt das Architekturmuster CORBA (Common Object Request Broker Architecture) dar. Es ermöglicht objektorientierte verteilte Kommunikation in heterogenen Umgebungen.

Ganz nach dem Konzept vom „Programmieren im Großen versus Programmieren im Kleinen" entwickelten sich diese Ideen zum Paradigma der serviceorientierten Architektur weiter.

> „We need languages for programming-in-the-small, i.e. languages not unlike the common programming languages of today, for writing modules. We also need a „module interconnection language" for knitting those modules together into an integrated whole and for providing an overview that formally records the intent of the programmer(s)." [DK75]

Dieses ist ebenso ein Programmierparadigma, welches auf der objektorientierten und komponenten-basierter Entwicklung aufsetzt wie auch eine Weiterentwicklung von heterogener verteilter Kommunikation in Systemen. Im „Kleinen" werden in SOA Komponenten, sogenannte Dienste, entwickelt, welche parallel entwickelt werden können und Teilaufgaben im System übernehmen. Im „Großen" erfolgt die Orchestrierung dieser Dienste. Auf dieser hohen Abstraktion werden die Geschäftsprozesse des Systems implementiert.

Dieser Blickwinkel hilft SOA als abstraktes Konzept zu sehen, für das es verschiedenste Implementierungen geben kann.

1.2 Definition von SOA

SOA stellt ein abstraktes Konzept einer Architektur dar welches sich mit dem Anbieten, Suchen und Nutzen von Diensten im Netz befasst. Diese Dienste werden plattformu-

nabhängig von Applikationen genutzt. Diese funktionale Zerlegung von Anwendungen erleichtert die prozessorientierte Betrachtungsweise und soll es ermöglichen komplexe Systeme zu entwickeln und diese überblickbar, skalierbar und wartbar zu halten.

Durch diesen generellen Ansatz finden sich in der Literatur jedoch unterschiedlichste Definitionen von SOA. Sie betrachten das Architekturkonzept aus verschiedenen Blickwinkeln und unterschiedlichen Graden der Abstraktion:

> **Definition 1:** „A service-oriented architecture (SOA) is a combination of consumers and services that collaborate, is supported by a managed set of capabilities, is guided by principles and is governed by supporting standards.“ [Bea07]

> **Definition 2:** „Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine Plattform und Sprachen unabhängige Nutzung und Wiederverwendung ermöglicht.“ [Bea10]

> **Definition 3:** „Zeitgemäße SOA bietet eine offene, erweiterbare, föderale, kombinierbare Architektur, die Service-Orientierung fördert und autonomen, Quality-of-Service-fähigen, unterschiedliche Hersteller unterstützenden, interoperablen, auffindbaren und potenziell wiederverwendbaren Services besteht, die als Web-Services implementiert werden. SOA kann zu einer Abstraktion von Geschäftslogik und Technologie führen und auf diese Weise eine lose Kopplung zwischen beiden Gebieten schaffen...“ [Erl05]

Diese Definitionen greifen die Kernmerkmale - Lose Kopplung, Verteiltheit, Prozessorientiertheit, aber auch fundamentale Eigenschaften wie Einfachheit und Standards - von SOA auf welche in Kapitel 1.6 „Merkmale einer Service orientierten Architektur“ genauer beleuchtet werden.

> **Definition 4:** „Eine SOA ist das Model eines Systems, welches vollständig aus autonomen Services aufgebaut ist, deren Interaktion über dasselbe öffentliche Protokoll abläuft und im Modell stets die drei Rollen Provider, Consumer und Broker vorhanden sind.“ [Mas10]

Das Suchen und Nutzen von Diensten lässt die drei Rollen in einer SOA ableiten, welche in Definition 4 auch konkret genannt werden: Provider, Consumer und Broker. Die Bezeichungen unterscheiden sich in der Literatur und treten z.B auch als Producer, Requestor und Repository auf. Übersetzt sind die Bezeichnungen Dienstanbieter, Dienstnutzer und Dienstverzeichnis (oder Vermittler) gängig und intuitiv verständlich.

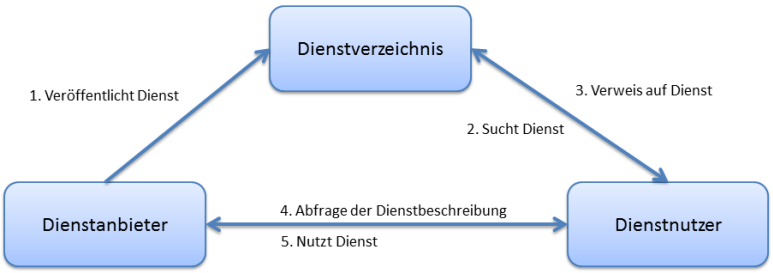

Abbildung 1.1: SOA Komponenten: Dienstanbieter, Dienstnutzer, Dienstverzeichnis

1.2.1 Dienst

Ein Dienst ist ein Programm oder eine Komponente, welche von anderen genutzt werden kann. Ob die Nutzung lokal oder entfernt stattfindet ist für den Nutzenden nicht von Bedeutung. Um dies zu ermöglichen muss eine Beschreibung des Dienstes in maschinenlesbarer Form vorliegen. Diese Schnittstelle verbirgt die Implementierung des Dienstes vor dem Dienstnutzer und setzt gleichzeitig das Prinzip des Information Hiding um.

> **Definition: Service - SOA Referenzmodell** „A service is a mechanism to enable access to a set of one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. A service is provided by one entity - the service provider - for use by others, but the eventual consumers of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by the provider.“ [MLM+06]

Die Beschreibung einer solchen Schnittstelle muss programmiersprachen- und implementierungsunabhängig geschehen um die Interoperabilität von Diensten zu gewährleisten. Solche Beschreibungssprachen sind z.B. IDL (Interface Definition Language), welche oft in Zusammenhang mit dem Architekturmuster CORBA verwendet wird oder WSDL (Web Service Description Language), welche sich im Bereich der Web Services durchgesetzt hat. Eine Dienstbeschreibung (Service Contract) umfasst die Funktionalität des Dienstes, wobei diese sehr stark an Signaturen von Funktionen in Programmiersprachen angelehnt ist und sie erlauben kaum Restriktionen oder gar höhere Semantik. Ebenfalls in die Dienstbeschreibung einfließen können Service Level Agreements, welche nicht funktionale Anforderungen an den Dienst, wie Antwortzeiten, Verfügbarkeit oder Sicherheitsmerkmale, darstellen.

1.2.2 Dienstanbieter

Der Dienstanbieter stellt eine Plattform zur Verfügung, über die er einen oder mehrere Dienste anbietet. Er ist somit auch für die Infrastruktur, den Betrieb und die Wartung dieser Plattform verantwortlich. Das bedeutet, er muss die in der Dienstbeschreibung festgelegten Service Level Agreements einhalten, auch dann wenn er die Dienste nicht selbst implementiert hat, sondern z.B Dienste anderer Anbieter nutzt und deren Funktionalität zu einem neuen Dienst kombiniert. Dies betrifft auch Service Level Agreements im Bereich Sicherheit und umfasst:

1. **Authentifizierung**
   Ist der Dienstnutzer derjenige, der er vorgibt zu sein?
2. **Autorisierung**
   Ist der Dienstnutzer berechtigt die angeforderte Funktionalität zu nutzen?

Der Dienstanbieter registriert die auf seiner Plattform laufenden Dienste im Dienstverzeichnis, damit sich von Dienstnutzern gefunden werden können.

1.2.3 Dienstverzeichnis

In der einfachsten Ausprägung registrieren sich alle Dienste bei einem zentralen Dienstverzeichnis, damit sie von Nutzern gefunden werden können. Nach [Bea10] stellt eine mögliche Ausbaustufe das Kaskadieren von Dienstverzeichnissen ähnlich des Domain Name Services dar. Solche Verzeichnisse haben eine baumartige Hierarchie und weisen eine unterschiedliche Datenbasis auf. Kann eine Anfrage nicht beantwortet werden wird diese an die nächst höhere Instanz weitergeleitet. Auch Caching von Antworten auf weitergeleitete Anfragen kann in Betracht gezogen werden. Eine weitere Möglichkeit stellt ein Dienstverzeichnis dar, welches, ähnlich wie Suchmaschinen im Internet, aktiv nach angebotenen Diensten sucht und diese katalogisiert. Eine automatische Kategorisierung von Diensten erweist sich jedoch als schwierig, wenn nicht unmöglich.

1.2.4 Dienstnutzer

Der Dienstnutzer hat eine lose Bindung zum gewünschten Dienst da diese Bindung nicht wie bei klassischen Remote Procedure Calls explizit kodiert ist, sondern der benötigte Dienst erst zur Laufzeit im Dienstverzeichnis gesucht wird. Ist diese Bindung hergestellt, agiert der Dienstnutzer wie ein traditioneller Klient einer Client-Server-Architektur.

> „In der Praxis wird diese lose Bindung oft aufgeweicht oder sogar aufgehoben, wenn sich Dienstnutzer und Dienstanbieter bekannt sind. In einem solchen Fall wird der Schritt über das Dienstverzeichnis oft übersprungen. Dies schont Ressourcen und verbessert die Laufzeit, reduziert aber die Flexibilität, da Änderungen nicht mehr nur über dieses Dienstverzeichnis ver-

breitet werden können. In den meisten Fällen ist dieser pragmatische Ansatz von Nachteil und auf Dauer ist der Verzicht auf die lose Bindung teuer.“ [Bea10]

Um die Netzwerke zu entlasten kann ein Serviceproxy zur Verfügung gestellt werden, welcher die Referenzen des Dienstanbieters sowie den Servicevertrag lokal speichert. Wird der gleiche Service öfters ausgeführt, muss der Dienstnutzer nicht jedesmal über das Dienstverzeichnis den Dienst anfragen. Alle Servicemethoden, die keine Daten vom Dienstanbieter benötigen, können so lokal auf dem Proxy ausgeführt werden. Mit einem Servicelease wird vereinbart, wie lange ein Servicevertrag und somit auch der Proxy gültig bleibt. Ist das Lease abgelaufen, muss der Dienstnutzer erneut eine Anfrage an das Dienstverzeichnis stellen. Der Zustand der Bindung wird über diesen Zeitraum aufrecht erhalten und ist unabdingbar für z.B. session-based Services, die diese Bindungsinformation benötigen. Des Weiteren reduziert die zeitliche Beschränkung der Gültigkeit eines Vertrags die Kopplung zwischen Anbieter und Nutzer, da ohne diese ein Nutzer für unbegrenzte Zeit an einen Service gebunden bleiben könnte. Das würde dem Kerngedanken von SOA, der losen Kopplung, widersprechen.

1.3 Serviceschichten

1.3.1 Entwicklung

Die Entwicklung von Softwarearchitekturprinzipien hat sich parallel zur Entwicklung der in Kapitel 1.1 „SOA als abstraktes Konzept“ angesprochen Programmierparadigmen vollzogen. Gleich wie in der Programmierung an sich wurde man sich der Notwendigkeit von mehr Abstraktion, Erweiterbarkeit und Veränderbarkeit erst nach und nach bewusst.

Die ersten großen Softwaresysteme waren monolithisch und ohne Bedacht auf heutige Architekturprinzipien entworfen. Das erste Prinzip war die sogenannte Schichten- oder Layerarchitektur, die in ihrer einfachsten Ausprägung eine Client-Server Architektur und generalisiert eine n-Tier Architektur darstellt. Schichten einer solchen Architektur implementieren Teilaspekte des zu lösenden Problems auf der jeweiligen Abstraktionsschicht. Sie bauen dabei aufeinander auf und nutzen dazu die Funktionalität der jeweiligen darunter liegenden Schicht. Eine meist genutzte Einteilung dieser Schichten umfasst:

1. **Benutzerschnittstelle**
   Dies stellt die Schicht zur Interaktion mit einem Benutzer dar und kann grafische Benutzeroberflächen, aber auch Hardware umfassen.

2. **Präsentationsschicht**
   Diese Schicht bereitet Informationen auf und stellt sie der Benutzerschnittstelle zur Verfügung.

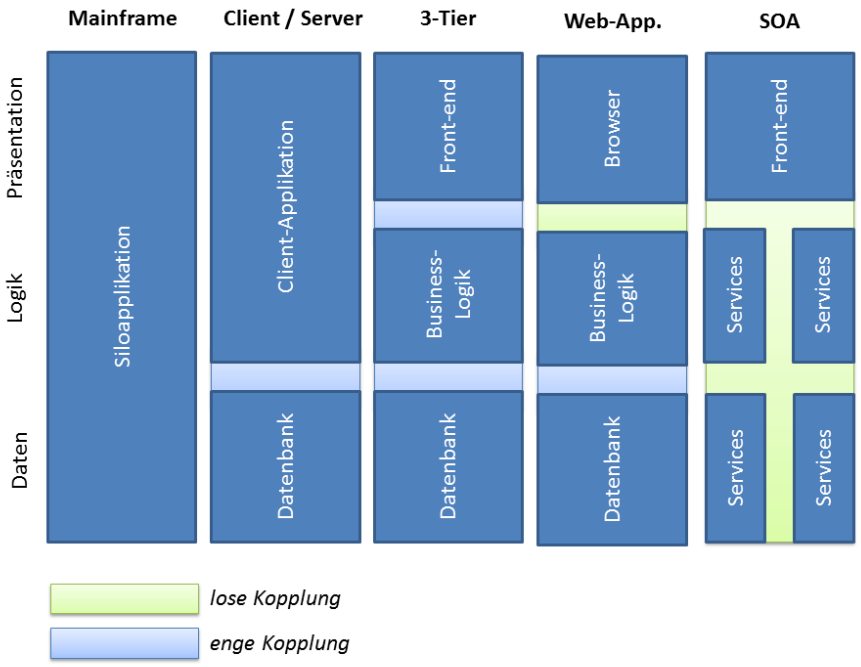

Abbildung 1.2: Evolution der Architektur (Überarbeitet aus [Mas10])

3. **Prozessschicht**
   Diese Schicht wird benötigt, wenn im System zeitübergreifende Prozesse, sogenannte Workflows, ablaufen. Dies ist aber bei weitem nicht in allen Systemen der Fall.

4. **Geschäftslogik**
   Modelliert domänenspezifische Objekte und Operationen auf diese.

5. **Datenzugriffsschicht**
   Diese stellt die Basisdaten für die Objekte der Geschäftslogik zur Verfügung.

Die Einführung von Schichten stellt ein wichtiges Abstraktionsmittel dar und wird auch für Dienste in einer SOA, wenn auch auf einer höheren Ebene, verwendet. In der Literatur [KBS04] [Erl05] [JB06] finden sich unterschiedlichste Einteilungen von Diensten in sogenannte Service-Layer, die sich im Kern aber kaum unterscheiden, nur mögliche Sublayer aufzeigen oder weglassen.

In [Mas10] werden Dienste in Service-Layer eingeteilt:

- **Business Services**
  Umfasst prozessorientierte Dienste, die auf hoher Abstraktion Geschäftsoperationen durchführen.

- **Business Neutral Services**
  Haben nichts mit dem Kerngeschäft zu tun, bieten aber domänenunabhängige komplexere Funktionalität, auf der Business Services aufbauen. Diese Dienste

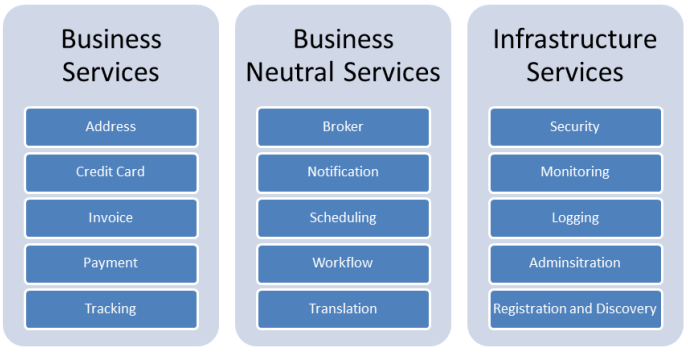

Abbildung 1.3: Serviceschichten und Anwendungsgebiete nach [Mas10]

sind am ehesten wiederverwendbar.

- **Infrastruktur Services**
  Umfasst grundlegende Komponenten die die nötige Infrastruktur zur Verfügung stellen. Streng genommen umfassen sie nicht die Anforderungen an Dienste, da sie keine fachliche Funktionalität bilden.

Diese Einteilung sehr intuitiv und passt gut auf die technischere Einteilung von [Jos09], die im Kapitel 1.3.2 „Klassifizierung von Diensten“ genauer beleuchtet wird.

1.3.2 Klassifizierung von Diensten

In [Jos09] werden Dienste nach drei Arten klassifiziert, wobei sich diese Klassifikation stark an [Erl05] anlehnt. Aus diesen werden unterschiedliche Ausbaustufen einer SOA aus [KBS04] abgeleitet. Diese Sichtweise ist nicht so systemorientiert wie die in Kapitel 1.3 „Serviceschichten“ vorgestellte Einteilung in Service-Layer, sondern eher domänenorientiert.

Basis-Services

Basis-Services kapseln eine fachliche Basisfunktionalität und stellen die erste Schicht über einem Back-end oder einer fachlichen Domäne dar und stellen diese Funktionalität höherwertigen Diensten zur Verfügung. Diese Dienste sollten kurzlaufend und zustandslos sein. Weiters können Basis-Services in Basis-Daten-Services und Basis-Logik-Services unterschieden werden.

Basis-Daten-Services
Basis-Daten-Services lesen oder schreiben Daten aus einem Backend. Dabei kapseln sie plattformspezifische Eigenschaften und sollten eine, möglichst minimale, fachliche Funktionalität anbieten. Für ihre Operation gilt das ACID-Prinzip:

Atomic	Ein Dienstaufruf ist entweder erfolgreich oder hat keinen Effekt.
Consistet	Der Dienstaufruf hinterlässt im Back-end einen gültigen, konsistenten Zustand.
Isolated	Während eines Aufrufs kann dieser nicht durch andere Aufrufe oder Veränderungen beeinflusst werden.
Durable	Ist ein schreibender Zugriff erfolgreich, so kann sein Effekt nicht ungeplant ungeschehen gemacht werden.

Basis-Logik-Services
Diese greifen nicht auf Daten zu, sondern repräsentieren einfache Geschäftsregeln. Daten werden ihnen übergeben und vom Aufruf ein dazugehöriges Ergebniss geliefert. Typischerweise wird es mehr Basis-Daten- als Basis-Logik-Dienste geben. Außerdem ist der Übergang fließend, da auch Logik-Dienste auf Daten aus einem Backend zugreifen können.

Composed-Service

Composed-Services werden aus Basis-Services oder anderen Composed-Services kombiniert. Sie dienen der Orchestrierung der Basisfunktionalität und werden deshalb oft als orchestrierte Services bezeichnet. Sie bilden aus Geschäftsprozesssicht Mikro-Flows aus den von den Basis-Services angebotenen Geschäftsoperationen. Gleich wie Basis-Services sind Composed-Services immer noch relativ kurzlaufend und zustandslos.

Composed-Services sind in der Regel für mehrere Backends verantwortlich und müssen für Operationen die Konsistenz in diesen Backends sicherstellen. Wenn ein Composed-Service nur für eine Backend verantwortlich ist, handelt es sich um einen so genannten Adapter-Service. Seine Aufgabe ist es dem Dienstnutzer eine andere Schnittstelle für den Basis-Service zur Verfügung zu stellen. Ein häufiges Anwendungsgebiet stellt hierbei die Erhaltung von Rückwärtskompatibilität zu früheren Servicesversionen dar. Aus Nutzersicht müssen die Operationen, die ein Composed-Service anbietet, ebenfalls dem ACID-Prinzip folgen. In der Praxis ist es jedoch schwierig oder oft auch unmöglich einen Transaktionsscope über Aufrufe in mehrere Backends auf zu spannen. Deshalb verwendet man anstatt Transaktionen und Two-Phase-Commit (2PC) das Prinzip der Kompensation.

Prozess-Services

In der letzten Ausbaustufe einer SOA kommen Prozess-Services hinzu. In [Erl05] wird der Begriff Prozess-Services und Composed-Services zusammengelegt. Nach [Jos09] und [KBS04] kann dies unter gewissen Gesichtspunkten sinnvoll sein. Um sich Schritt für Schritt einer SOA Vollausbaustufe anzunähern ist diese Aufteilung aber zu bevorzugen.

Prozess-Services sind Dienste die Geschäftsprozesse verwalten. Diese erstrecken sich meist über einen längeren Zeitraum und sind zustandsbehaftet. Des Weiteren können

sie durch Benutzerinteraktion unterbrochen werden. Die logische Session ist nicht mit einem einzigen Frontend verbunden. So kann der Prozess selbsttätig oder über verschiedene Forntends von Zustand zu Zustand wechseln. Ob der Zustand eines solchen Dienstes im Dienst oder im Backend gehalten wird, ist eine Designentscheidung, die für konkrete Problemstellungen unterschiedlich ausfallen kann. Generell kann man sagen, dass die Wichtigkeit der Zustandsdaten über deren Lokalisierung entscheidet. Nach [Jos09] ist es angemessen den Zustand im Backend zu verwalten, wenn dieser „juristisch relevant“ ist, also auch nach längeren Zeiträumen auffindbar und nachweisbar sein muss. Ist dies nicht der Fall, sind zustandsbehaftete Dienste eine angemessene Lösung.

1.4 Enterprise Service Bus

1.4.1 Aufgaben

Die Aufgabe eines ESB besteht darin Dienste einfach systemübergreifend aufrufen zu können. Das bedeutet Interoperabilität zwischen Dienstnutzer und Dienstanbieter herzustellen. Die Umsetzungen eines Enterprise Service Bus können konzeptionell wie technisch sehr unterschiedlich sein und die Meinungen was ein ESB leisten muss gehen in der Literatur auseinander. In [Jos09] werden folgende allgemeine Aufgaben eines ESB identifiziert:

- Konnektivität herstellen
- Daten transformieren
- Intelligent routen
- Mit Sicherheitsaspekten umgehen
- Mit Aspekten der Zuverlässigkeit umgehen
- Services verwalten
- Möglichkeiten zum Überwachen, Protokollieren und Debuggen bereitstellen.

Die wichtigste Aufgabe hierbei besteht darin die Interoperabilität zwischen verschiedenen Plattformen, Hardware und Programmiersprachen herzustellen. Um dies zu erreichen müssen die Aufgaben der Konnektivität, Datentransformation und des Routings hinreichend gelöst werden. In [Cha04] wird in Bezug auf Datentransformation den Begriff des Impedanz-Unterschieds verwendet.

> „Datentransformation ist ein inhärenter Teil des Busses in einem ESB-Deployment. Transformations-Dienstleistungen, die auf die einzelnen am Bus angeschlossenen Anwendungen spezialisiert sind , findet man überall und kann man von überall aufrufen. Da Datentransformation ein derartig integrierter Bestandteil eines ESB ist, kann man es auch so betrach-

ten, dass ein ESB den Impedanz-Unterschied zwischen Anwendungen ausgleicht.“ [Cha04]

Dieser Begriff kommt ursprünglich aus der Elektronik und steht in der IT für den konzeptionellen Unterschied von Systemen wie z.B. den von objektorientierten und relationalen Datenstrukturen. Zur Kernaufgabe des ESB gehört also, diesen Unterschied zwischen seinen Anwendungen (Diensten), welche zu unterschiedlichen Zeiten von unterschiedlichen Teams für unterschiedliche Plattformen entwickelt wurden, aufzuheben. Um Interoperabilität zu erreichen ist neben Datentransformation auch Routing von Nachrichten notwendig. Dies kann trivial sein, hängt aber von den verwendeten Protokollen sowie Message-Exchanges-Patterns ab. Die Komplexität weiter erhöhen können Systeme, die Prioritäten und Geltungsbereiche für Nachrichten betrachten. Schon diese Grundaufgaben stellen besonders in Hinblick auf Performance eine Herausforderung dar. Je zeitkritischer die implementierte Anwendung umso mehr muss der zeitliche Aufwand für Datentransformationen und Routing (hier besonders Content-Based-Routing) berücksichtigt werden.

1.4.2 Erweiterungen

Ein Streitpunkt in der Literatur besteht darin, ob die Genannten drei Aufgaben nun die Kernaspekte eines ESB sind und die übrigen lediglich als Erweiterungen zu betrachten sind oder ob diese gar noch erweitert werden sollen. Ein Beispiel für eine solche Erweiterung stellt eine die Integration einer BPEL-Engine in die Grundaufgaben eines ESBs dar, welche zum Überwachen und Verfolgen von Geschäftsprozessen dient. Dieses sogenannte Business-Activity-Monitoring (BAM) ist ein wichtiges Argument für einen intellegenten ESB und wird oft auch als Business-Case für SOA als Ganzes verwendet. Duch das Überwachen der fachlichen Aktivitäten auf dem ESB soll man in die Lage versetzt werden Tendenzen und Störungen in den Geschäftsprozessen zur Laufzeit zu erkennen und darauf reagieren zu können.

1.4.3 heterogener ESB

Ein ESB selbst muss aber nicht zwingend homogen sein. Obwohl dies wünschenswert wäre, kann dies oft nicht erreicht werden. Die beteiligten Protokolle entwickeln sich ständig weiter, SOA-Landschaften wachsen zusammen und verändern sich. Auch wenn es gelingt sich firmenintern zu einigen werden Drittanbieter von diesen Konventionen abweichen.

Bei der Implementierung einer SOA sollte man sich also auf Heterogenität von vornherein einstellen. Darum wird in [Jos09] auch die Unabhängigkeit der Dienstschnittstellen von der Infrastruktur gefordert:

„Im Idealfall sollte sichergestellt werden, dasss die Details der Infrastruktur keinen Einfluss auf die Schnittstellen der Anbieter und Nutzer haben. Bei

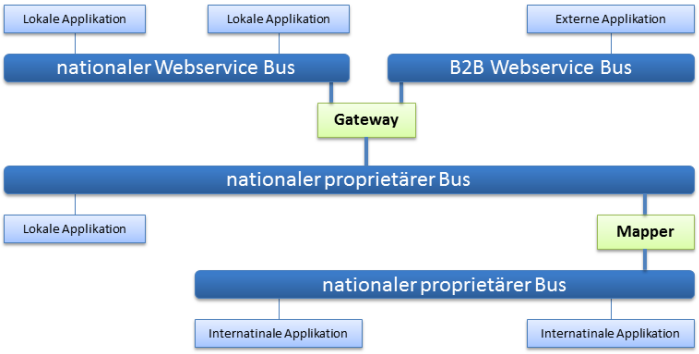

Abbildung 1.4: Beispiel eines heterogenen ESB (Überarbeitet aus [Jos09])

> einem Wechsel des ESB sollte das fachliche API der Services erhalten bleiben. Lediglich das Mapping dieses APIs auf das Protokoll des ESB sollte angepasst werden.“ [Jos09]

Ein ESB stellt also die Infrastruktur einer SOA-Landschaft zur Verfügung, indem er Interoperabilität zwischen den Anbietern und Nutzern von Diensten herstellt. Implementierungen eines ESB können sich weitgehend unterscheiden. Er kann einfach durch ein Protokoll realisiert werden, das auf standardisierten Netzwerkprotokollen aufsetzt oder aus einer Vielzahl von Tools und Softwarekomponenten, die zentral oder dezentral von allen Beteiligten genutzt werden, bestehen. Des weiteren kann er um weitere Aufgaben wie Sicherheit, Monitoring oder komplexere Aufgaben wie Geschäftsprozessüberwachen erweitert werden.

1.5 Ausbaustufen einer SOA

Aus der Klassifikation von Diensten in Basis-, Composed- und Prozess-Services lässt sich die in [KBS04] und [Jos09] definierten Ausbaustufen einer serviceorientierten Architektur ableiten.

1.5.1 Fundamentale SOA

Die erste Ausbaustufe einer SOA stellt die fundamentale SOA dar. Dienstnutzer greifen über den Enterprise-Service-Bus auf die fachlichen Schnittstellen mehrerer Backends, den Basis-Services, zu. Jedes dieser Backends soll eine klare Rolle besitzen, die sich in der Organisationsstruktur des Unternehmens und der Verantwortlichen widerspiegelt. Beispiele hierfür wären Kundenverwaltung, Gebäudeverwaltung, Buchhaltung oder Messdatenzugriff. In der Praxis sind die Systemlandschaften gewachsen und Subsysteme oft stark miteinander verwoben. Wie soche Systeme zu einer SOA-Landschaft werden können wird in Kapitel 1.8 „Einführen von SOA“ ausführlicher behandelt.

Grundsätzlich fällt eine Einführung Unternehmen mit höherem Business/IT Alignment, also der Abstimmung zwischen IT-Strategie und fachlicher IT, leichter.

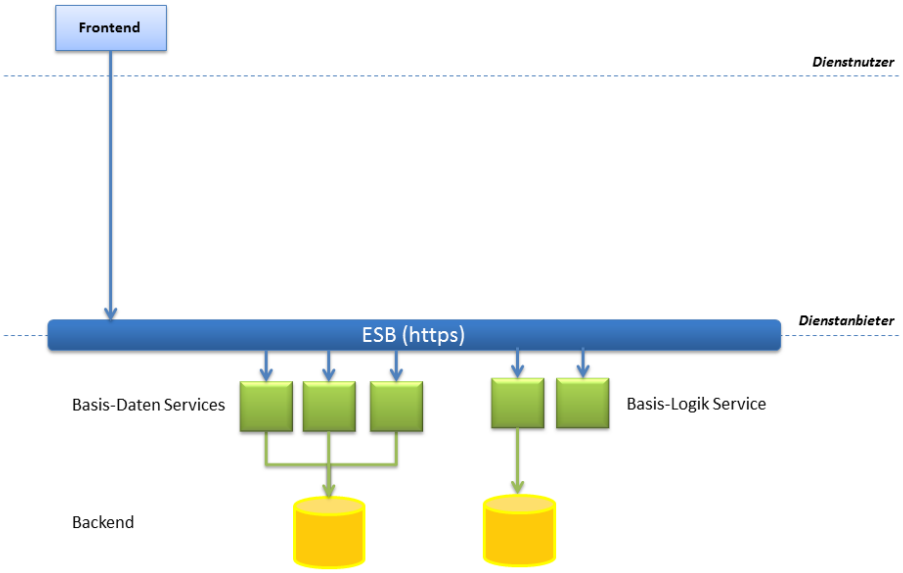

Abbildung 1.5: Fundamentale SOA

Wie Abbildung 1.8.3 zeigt, kapseln Basis-Services mehrere Backends und es kann natürlich mehrere Dienste für ein Backend geben. Basis-Services greifen jedoch nie auf mehrere Backends zu. Auch kümmern sie sich nicht im Inkonsistenzen zwischen Backends. Ein Beispiel für Inkonsistenzen, das in der Praxis in gewachsenen Systemen oft auftritt, wären redundant gehaltene Daten in unterschiedlichen Backends. Konkret könnte sowohl das Liefer- wie auch das Buchhaltungssystem die Kundenadresse abgelegt haben. Für beide Backends gibt es in den jeweiligen Basis-Services eine Operation zum Ändern dieser Adresse. In einer fundamentalen SOA ist es die Aufgabe der Dienstnutzer mit solchen Inkonsistenzen umzugehen, also bei einer Adressänderung beide Dienste aufzurufen. Dienstnutzer sind können in diesem Szenario Clients, Front-Ends oder Batch-Programme sein.

1.5.2 Föderierte SOA

Föderierte SOA, die nach [KBS04] auch Netzwerk-SOA genannt wird, erweitern die fundamentale SOA um Composed-Services. Diese übernehmen dei Aufgabe Konsistenz in den unterscheidlichen Backends herzustellen. Dazu müssen sie wissen welche Backends informiert werden müssen und wie sich die Operation auf die angebotenen Basis-Services abbilden lässt.

Composed-Services müssen außerdem im Fehlerfall geeignet reagieren. Wie in Kapitel 1.3.2 „Basis-Services“ gefordert müssen das ACID-Prinzip einhalten. Wie schon erwähnt ist es schwierig Transaktionsscopes über mehrere Backends auf zu spannen. Kommen zeitlich länger laufende Prozesse hinzu, kann dies gar unmöglich werden. Statt

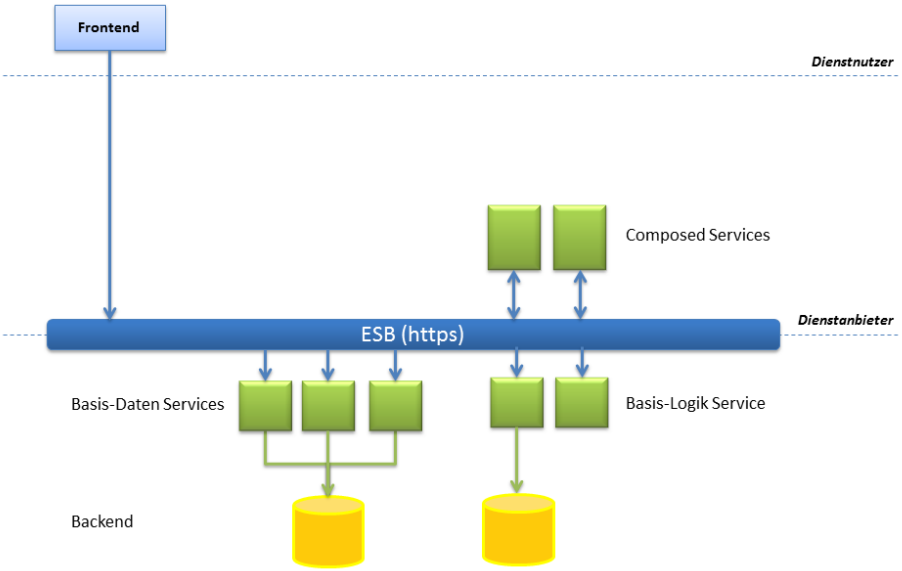

Abbildung 1.6: Föderierte SOA

Transaktionen wird das Prinzip der Kompensation angewandt. Dazu müssen die Basis-Services jedoch Umkehrfunktionalität zur Verfügung stellen. Eine weitere Möglichkeit wäre, im Fehlerfall einen dritten Dienst zu informieren, der die Kompensation ausführt, oder abstrakter gesehen den Fehler an einen Fehlerarbeitsplatz weiterzuleiten. Somit könnte das in Abschnitt 1.5.1 angeführte Beispiel mit einem Composed-Service gelöst werden, dessen Implementierung wie folgt aussehen könnte:

Listing 1.1: Beispiel: Composed-Service (Pseudocode)

```
try
{
    try
    {
        alteAdresse = LiefersystemServiceProxy.LeseAdresse()
        LiefersystemServiceProxy.ÄndereAdresse(neueAdresse)
    }
    catch(Fehler) //Backend trotzdem konsistent
    {
        return "Adresse konnte nicht geändert werden."
    }

    try
    {
        BuchhaltungssystemServiceProxy.ÄnderAdresse(neueAdresse)
    }
    catch(Fehler) //Kompensation
    {
        LierfersystemServiceProxy.ÄndereAdresse(alteAdresse)
        return "Adresse konnte nicht geändert werden."
    }
}
catch(Fehler) //Kompensation fehlgeschlagen
{
    return "Schwerer Fehler."
}
```

Kann die erste Teiloperation nicht erfolgreich durchgeführt werden, befinden sich beide Backends noch immer in einem konsistenten Zustand. Ist die Änderung jedoch im ersten Backend erfolgt und schlägt im zweiten fehl, muss dieser Fehler kompensiert werden. Schlägt auch die Kompensation fehl, kommt es zu einem schweren Ausnahmefehler. Aus Gründen der Nachverfolgbarkeit sollten Ereignisse wie die Kompensation eines

Fehlers im System aufgezeichnet werden. Da alle Dienste einem Dienstnutzer über den Enterprise Service Bus zugänglich sind, kann dieser auch die Basis-Dienste nutzen. Dies ist auch erwünscht, wenn der Nutzer nur eine Basisfunktionalität benötigt. Wie das oben gezeigte Beispiel deutlich macht, ist er dadurch aber in der Lage Inkonsistenzen im Backend zu erzeugen. Darum ist es oft notwendig Basis-Services vor Endnutzern zu verstecken.

1.5.3 Prozessfähige SOA

Prozess-Services erlauben nun Geschäftsprozesse im System abzubilden und diese zu verwalten. Sie können über mehre Subsystem laufen, von unterschiedlichen Frontends bedient werden und durch menschliche Interaktion angestoßen und unterbrochen werden.

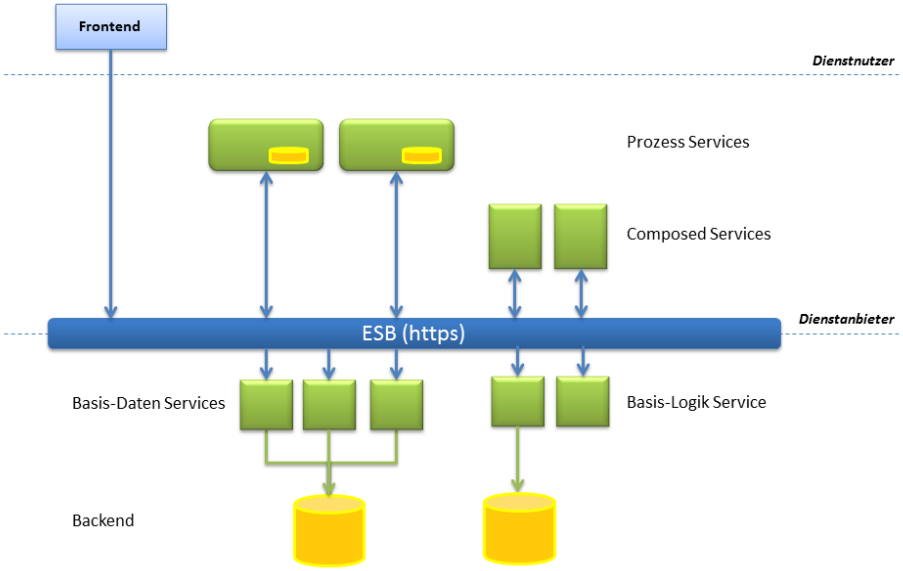

Abbildung 1.7: Prozessfähige SOA

Ein Prozess führt zwangsweise einen Zustand in ein System ein. Im einfachsten Fall ist es die Information, in welchem Schritt des Prozesses sich dieser, für welchen Akteur, befindet. In der Realität kommt jedoch eine Vielzahl von Prozessdaten hinzu. Wo dieser Zustand im System abgeleitet werden soll, wurde schon in Abbschnitt 1.3.2 „Prozess-Services“ diskutiert. Auf dieser Stufe von SOA werden BPEL-Engines interessant. BPEL steht für Business Process Execution Language, ein Standard zur Beschreibung von Geschäftsprozessen.

> **Definition: BPEL** „Die WS-Business Process Execution Language (BPEL) ist eine XML-basierte Sprache zur Beschreibung von Geschäftsprozessen, deren einzelne Aktivitäten durch Webservices implementiert sind. Die im Jahr 2002 von IBM, BEA Systems und Microsoft

eingeführte Sprache wird dabei zur Beschreibung von so genannten Webserviceorchestrierungen verwendet. Die Beschreibung selbst wird ebenfalls in Form eines Webservice bereitgestellt und kann als ein solcher verwendet werden. Durch die Abstraktion mittels BPEL kann die Schnittstelle eines Webservice, der die an einem Prozess beteiligten Webservices steuert, beschrieben werden - beispielsweise in welcher Reihenfolge Nachrichten eintreffen müssen. WS-BPEL ist Teil der sogenannten WS-*-Spezifikationen.“ [Wik10e]

Für BPEL gibt es unterschiedliche kommerzielle wie auch freie Implementierungen wobei als Innovationstreiber hier IBM, Microsoft und BEA Systems zu nennen sind. Die prominentesten davon sind etwa jBPM als Bestandteil des JBOSS Applikationsservers oder seit neuerem die Microsoft Windows Workflow Foundation, welche im .NET 3.0 Framework enthalten ist.

Diese Engines beinhalten meist grafische Editoren, in denen Prozesse aus einzelnen Aktivitäten modelliert werden können. Hierbei kommen folgende Aktivitäten als Sprachbestandteile zum Einsatz [DJ07].

Basic Activities bilden die Grundaktivitäten und umfassen:

assign	Das Zuweisen von Variablen.
invoke	Synchroner oder asynchroner Aufruf eines Dienstes.
receive/reply	Anbieten eines synchronen oder asynchronen Dienstes.
throw	Signalisieren eines Fehlers, welcher durch Fehlerbehandlungen aufgefangen werden kann.
wait	Warten auf einen Zeitpunkt oder für eine Zeitspanne.
empty	Keine Aktivität.

Structured Activities beinhalten andere Aktivitäten und lassen so die rekursive Komposition von komplexen Prozessen zu.

sequence	Aktivitäten sequentiell abgearbeitet.
while	Ausführen von Aktivitäten, solange eine boolesche Bedingung erfüllt ist.
switch	Bedingte Ausführung von Aktivitäten.
flow	Die Aktivitäten werde parallel oder in beliebiger Reihenfolge ausgeführt.
pick	Aus Prozesssicht nicht-deterministische Wahl durch externe Ereignisse.

Mithilfe sogenannter Scopes können Aktivitäten zu einer transaktionalen Einheit zusammengefasst werden. Einem Scope kann einen Fault-Handler, Event-Handler und Compensation-Handler (siehe Kompensation in Abbschitt 1.5.2 „Föderierte SOA“) besitzen.

1.6 Merkmale einer Service orientierten Architektur

Die Grundbausteine einer SOA - Einfachheit, Sicherheit und Standards - sind eigentlich notwendige Voraussetzungen, die eine Akzeptanz dieser Architektur erst ermöglichen:

- **Einfachheit**
  Da Dienste Aufgabenbereiche kapseln, wird es möglich diese in verschiedenen Umgebungen wiederzuverwenden. Ein einzelner Dienst bleibt in überschaubarer Komplexität, erst das Zusammenspiel mehrerer Dienste spiegelt vollständige Geschäftsprozesse wider.
- **Sicherheit**
  Gerade in Verteilten Umgebungen ist Sicherheit ein wichtiges Thema. Eine Umsetzung von SOA muss Authentifizierung und Autorisierung der Kollaborateure im System berücksichtigen.
- **Standards**
  Um einen Dienst nutzen zu können muss seine Schnittstelle in maschinenlesbarer Form beschrieben sein. Dazu ist eine wichtige Voraussetzung, dass offene Standards genutzt werden, damit der Nutzer den Dienst eines unbekannten Anbieters auch verstehen kann.

Auf diesen Punkten bauen die Kernaspekte von SOA auf:

- **Lose Kopplung**
  Dienste werden von Nutzern, welche selbst wieder Dienste sein können, erst bei Bedarf dynamisch gesucht und eingebunden. Es handelt sich also um eine Bindung zur Laufzeit und unterscheidet sich so stark von RPC.
- **Verzeichnisdienst**
  Dienste registrieren sich bei einem Verzeichnisdienst, welcher das Suchen nach Methoden gestattet, die von einer Anwendung gerade benötigt werden.
- **Prozessorientiert**
  Aufgrund der flexiblen Architektur und losen Kopplung bieten sich Service orientierte Architekturen an, Prozessabläufe abzubilden. Die Architektur stellt sicher, dass auf Änderungen in diesen Prozessen besser reagiert werden kann.

1.7 Qualitätsattribute

Qualitätsattribute sind Eigenschaften, die eine Architektur vorantreiben, sie für eine spezielles Anwendungsgebiet interessant machen oder aber auch ausschließen können. Wie gut eine Architektur in einem Qualitätsattribut abschneidet, hat somit Auswirkungen auf die Implementierung. Die in [Mas10] vorgestellten Qualitätsattribute haben für Service orientierte Architekturen folgende Bewertung:

Attribut	Bewertung	Attribut	Bewertung
Interoperabilität	gut	Erweiterbarkeit	gut
Zuverlässigkeit	neutral	Adaptierbarkeit	neutral
Verfügbarkeit	neutral	Testbarkeit	schlecht
Usability	neutral	Nachverfolgbarkeit	schlecht
Sicherheit	schlecht	Betrieb- und Einsetzbarkeit	neutral
Performance	schlecht	Veränderbarkeit	gut
Skalierbarkeit	neutral		

Tabelle 1.1: Qualitätsattribute und ihre Bewertung für SOA [Mas10]

Interoperabilität

Interoperabilität ist eines der Hauptargumente für SOA und kommt vor allem in Integrationsszenarien stark zum Tragen. Interoperabilität ist die Fähigkeit, dass unterschiedliche Systeme (hier Dienste) Informationen gemeinsam nutzen und standardisiert semantisch interpretieren können. Um das Versprechen der Interoperabilität halten zu können verlassen sich z.B. Webservices Implementierungen von SOA auf die Standards WSDL (Web Service Description Language) und SOAP (Simple Object Access Protokol) zur Kommunikation zwischen Diensten. Dies ist nicht zwingend notwendig und eigene Lösungen sind möglich. Setzt man jedoch hier nicht auf einen breit unterstützten Standard, verabschiedet man sich von der Interoperabilität.

Zuverlässigkeit

Zuverlässigkeit ist die Fähigkeit eines Systems für lange Zeit aktiv zu sein. In einer SOA sind die Komponenten, für welche Zuverlässigkeit gefordert wird, das Dienstverzeichnis und die Dienste an sich, sowie der Nachrichtenaustausch zwischen Diensten. Da Dienste über Rechner und Netzwerkgrenzen verteilt sein können, kommt diesem Punkt besondere Bedeutung zu.

Verfügbarkeit

Die Verfügbarkeit bestimmt, wie sehr ein System erreichbar ist, wenn es benötigt wird. Die Dienste einer SOA wird das Maß an Verfügbarkeit im Servicevertrag in Form eines SLAs (Service Level Agreement) festgelegten und somit dieses Qualitätsattribut zu einem fixen Bestandteil der Nutzung gemacht. Grundlegend ist die Verfügbarkeit einer SOA aus architektonischer Sicht neutral zu bewerten, kann aber durch Einsatz von Technologien verbessert werden. Es können redundante Dienstinstanzen im System vorgesehen werden oder Protokolle wie Message Queue eingesetzt werden um Nachrichten zeitlich zu erhalten.

Nutzbarkeit

Die Nutzbarkeit bestimmt die Qualität der Nutzererfahrung. Innerhalb einer SOA betrifft dies den Umgang mit Diensten und nicht so stark die Ergonomie einer Benutzeroberfläche. Wichtig sind hier das Design der Dienstschnittstelle. Übersichtlichkeit und intuitives Verständnis der Schnittstelle sowie die richtige Datengranularität sind für die Nutzung wichtig. Dies betrifft auch das Dienstverzeichnis, welches es erlaubt Dienste ausfindig zu machen und diese zu binden. Zur Nutzererfahrung gehört auch, wie sich das System verhält, sollten Dienste temporär nicht zur Verfügung stehen. Ein Ausfall des Systems wäre hier inakzeptabel.

Sicherheit

Informationen stellen in jedem System einen Wert da den es zu schützen gilt. Eine Verfehlung in diesem Bereich kann kann schwerwiegende Folgen haben. Man denke hier z.B. an die Domänen Gesundheitswesen, Flugsicherheit oder Justiz. Da eine SOA zu einem offenen, verteiltem System führt ist die Sicherheit hier eine besondere Herausforderung. Es gilt Prozesse und deren Informationen in folgenden Kontexten zu schützen:

- **Vertraulichkeit**
  Informationen sollen nur einer autorisierten Entität (Person oder System) zur Verfügung gestellt werden. Dies schließt auch die Authentifizierung dieser Entität mit ein.
- **Integrität**
  Informationen müssen korrekt und vollständig verarbeitet und übermittelt werden.

Sicherheitsaspekte werden in der Praxis oft nur sekundär zur Funktionalität des Gesamtsystems gesehen und somit sind Geschäftsprozesse oft einem hohen Risiko ausgesetzt.

Performance

Im Punkt Performance schneiden serviceorientierte Architekturen aus vielerlei Gründen sehr schlecht ab. Dienste sind stark verteilt, was viel Kommunikation über Netzwerke bedeutet. Die für die Bindung notwendige Anfrage bedeutet einen nicht zu vernachlässigenden Performanceoverhead. Die Kaskadierung von Servicelayern fügt eine weitere Indirektion hinzu, da ein Service für die Durchführung seiner Aufgabe mehrere Dienste einer darunter liegenden logischen Schicht benötigt.

Daten müssen bei jedem Aufruf zeitintensiv verpackt und entpackt werden (Marshalling). Verwendet man Web Services als grundlegende Technologie (was zur Zeit den Regelfall darstellt) ist das Parsen und Validieren der XML Dokumente wie Service

Contract und serialisierte Daten ein nicht zu unterschätzender Performanceverlust. Echtzeitanwendungen sind somit praktisch auszuschließen da Netzwerke keine deterministischen Zeiten garantieren und die Zahl der Indirektionen bei Dienstaufrufen oft nicht vorhersehbar ist.

Skalierbarkeit

Die Skalierbarkeit einer SOA hängt weniger davon ab wie sie implementiert ist. Grundsätzlich beeinflusst die Trennung von Interface und Implementierung (Implementierungstransparenz) die Skalierbarkeit positiv, ausschlaggebend ist aber die Plattform auf der die Dienste laufen (siehe Dienstanbieter). Diese kann es z.B. erlauben Instanzen des selben Dienstes transparent auf mehrere Rechner zu verteilen und die Ressourcen im Subsystem optimal auszunutzen.

Erweiterbarkeit

Erweiterbarkeit ist ein Qualitätskriterium, welches serviceorientierte Architekturen sehr gut erfüllen können. Darunter versteht man Systemen neue Funktionalität hinzuzufügen, ohne dass das Gesamtsystem beeinflusst wird. Im Kontext von Diensten bedeutet das:

- **Hinzufügen von Services**
  Neue Services können einfach hinzugefügt werden und sie beeinflussen das Gesamtsystem nur, wenn sie explizit verwendend werden. Durch die dynamsiche Bindung von Services kann aber auch neue Funktionalität eingebunden werden ohne Änderungen vornehmen zu müssen.

- **Verändern von Services**
  Wird das bestehende Interface nicht verändert, hat dies keine Auswirkungen auf das Gesamtsystem. Diese Trennung von Schnittstelle und Implementierung ist ja ein Kernaspekt von SOA. Rein additive Erweiterungen können leicht durch ein neues Interface realisiert werden, wobei es kompatibel zum alten bleibt.

- **Veränderungen von Interfaces**
  Eine solche Veränderung „bricht“ das Interface und somit den Servicevertrag. Alle Systeme, die den betroffenen Dienst nutzen, müssen diese Änderung kompensieren. Solche Veränderungen sind soweit wie möglich zu vermeiden, da sie sehr aufwändig sind und unerwünschte Seiteneffekte mit sich ziehen können.

Wie bereits oben erwähnt, ist die Erweiterbarkeit einer SOA solange gewährleistet, wie die Schnittstellen ihrer Dienste stabil bleiben. Dies erfordert gute Designarbeit bei dem Entwurf dieser Schnittstellen.

Adaptierbarkeit

Hierunter wird verstanden wie gut sich ein System an Veränderungen der fachlichen Anforderungen anpassen kann. Eine SOA ermöglicht es leicht neue Services zu schaffen und bereits bestehende durch die lose Kopplung neu zu kombinieren und sich so auf Veränderung der Geschäftsprozesse einzustellen. Auch Übertragung von Diensten in eine andere Domäne kann erfolgreich sein, sofern der funktionale Deckungsgrad und die Granularität der Services dies zulassen.

Testbarkeit

Die Eigenschaften einer serviceorientieren Architektur können das Testen aufwendig machen. Durch die Verteiltheit der Dienste im Netz kann es zu Timingproblemen sowie Übertragungsproblemen kommen welche in Testszenarien nur schwer reproduzierbar sind. Somit kann die Implementierung eines Dienstes noch mit Unit-Tests getestet werden, Integrations- und Systemtest sind jedoch schwierig und es gibt noch keine standardisierten Vorgehensweisen dafür.

Nachverfolgbarkeit

Die Nachverfolgbarkeit ist schwer zu gewährleisten, insbesondere wenn externe Dienste eingebunden werden. In der Praxis bieten Dienst-Hosting-Plattformen von Dienstanbietern oft Funktionalität zur Nachverfolgbarkeit der Dienstaufrufe, diese kooperieren jedoch nicht mit Plattformen anderer Dienstanbieter.

Veränderbarkeit

Wie für die Erweiterbarkeit gilt, dass SOA ein hohes Maß an Veränderbarkeit verspricht, solange die Serviceinterfaces stabil bleiben. Werden diese aber „gebrochen“, wird Veränderung schnell problematisch.

1.8 Einführen von SOA

Eine service orientierte Architektur wird im Regelfall auf bereits existierende Systeme angewandt. Diese Einführung stellt eine Organisation weniger vor technische als vor organisatorische Herausforderungen. Der IST-Zustand des Systems muss ermittelt werden und die Dienste aus den Geschäftsprozessen abgeleitet werden.

> „Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organizations communication structure.“ *Conways Gesetz* [Con86]

Meist unterliegen die Subsysteme einer Organisation anderen Verantwortlichen die in die Einführung eingebunden werden und diese auch unterstützen müssen. In [Mel10] wird eine Einführung in zwei Schritten vorgeschlagen:

1.8.1 Ausgangszustand

Den IST-Zustand stellt in der Regel eine gewachsene Systemlandschaft dar, die aus einer Reihe von Systemen besteht. Jedes dieser Systeme besitzt seine eigene Datenbasis, internen Funktionen und Workflows. Zwischen diesen Systemen bestehen jeweils eigene eins-zu-eins Verbindungen die eine hohe Komplexität aufweisen und aufwändig gewartet werden müssen. Sogar systemübergreifende Abläufe sind oft fest kodiert. Somit wird jede Prozessänderung schwer umsetzbar, teuer und möglicherweise mit nicht vorherzusehenden Seiteneffekten einhergehen.

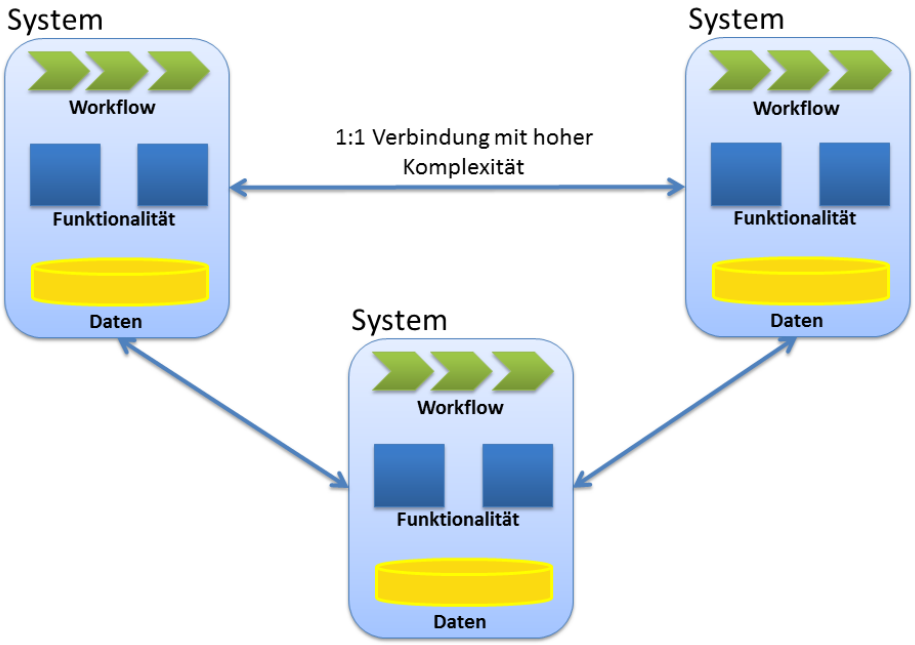

Abbildung 1.8: Ausgangszustand: Verwobene Systeme

Die beteiligten IT-Verantwortlichen denken in ihren Systemen und gesamte Zusammenhänge werden oft nicht wahrgenommen. Kaum jemand kann nur annähernd das Gesamtsystem überblicken.

1.8.2 Das Anbieten von Diensten

Ziel des ersten Schritts ist es, dass die bestehende Funktionalität der Systeme, die bereits von anderen Systemen genutzt wird, als Dienste angeboten werden. In diesem Schritt sind oft nur wenige Änderungen an den Anwendungen nötig jedoch muss viel über das Gesamtsystem gelernt und dokumentiert werden.

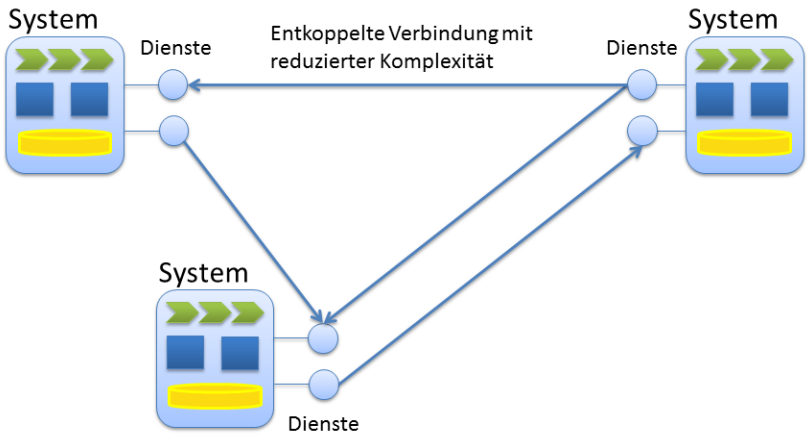

Abbildung 1.9: Systeme bieten Dienste an

Es wird dokumentiert welche Anwendungen auf welchem Weg welche Dienste anbieten. Es muss klar werden wo welcher Teil eines Workflows implementiert ist und die Abhängigkeiten der Systeme kennen. Durch das Anbieten von Diensten werden erreicht:

- Klare (und dokumentierte) Aufteilung der Zuständigkeiten
- Weniger Verbindungen unter den Systemen
- Niedrigere Komplexität dieser Verbindungen
- Erhöhte Flexibilität

Dieser Schritt ist beendet wenn es keine unbekannten entfernten Aufrufe mehr gibt. Die beteiligten Systemverantwortlichen denken jetzt zwar noch immer in ihren Systemen aber auch in den Diensten die sie dem Gesamtsystem anbieten.

1.8.3 Standardisierte Dienste

Im zweiten Schritt müssen nun die gefundenen Dienste standardisiert werden. Dazu müssen die systemübergreifenden Prozesse erfasst und verstanden werden, und in weiterer Folge, wie dafür die dahinter liegenden Dienste genutzt werden.

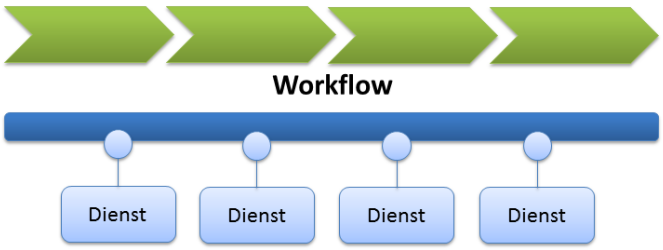

Abbildung 1.10: Standardisierte Dienste

Dienste kommunizieren über standardisierte Weise wie z.B. über einen ESB (Enterprise Service Bus). Durch den hohen Grad dieser Standardisierung sind nun einzelne Dienste einfacher auszutauschen und das System kann flexibler auf Prozessänderungen reagieren.

Eine Vorgehensweise, die sowohl [KBS04] als auch [Jos09] vorschlägt und sich am deutlichsten in den vier „P“ - „People, Pilot, Plan and Proceed“ aus [EP06] wiederfindet, ist, dass jedes Unternehmen mit einem überschaubaren Pilotprojekt sich der SOA Thematik nähern sollte. Es gilt also erst die beteiligten Personen mit den SOA Prinzipien und Technologien vertaut zu machen und die Konzepte in einem Pilotprojekt zu erproben. Danach bedarf es einer SOA-Strategie für das gesamte Unternehmen, die in alle zukünftige Projekte integriert werden muss.

1.9 Häufige Fehler bei der Einführung - SOA Antipatterns

„Today's hot solution that can be tomorrow's antipattern.“

Design-Patterns bieten erfahrungsgestützte Lösungen für wiederkehrende Probleme. In der Domäne der Softwareentwicklung erreichten sie durch Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides - auch die „Gang of Four“ [EG94] genannt - Bekanntheit.

> „A pattern is defined as a „generalized, named problem-to-solution mapping.“ It captures a successful solution to a repeating problem in a particular context.“ [JA05]

Designmuster können Softwaredesign, Topologie oder auch Entwicklungsprozesse umfassen und werden meist textuell beschreiben wobei man das Muster in Name, Beschreibung des Problems, Beschreibung der Lösung und Mögliche Konsequenzen gliedert.

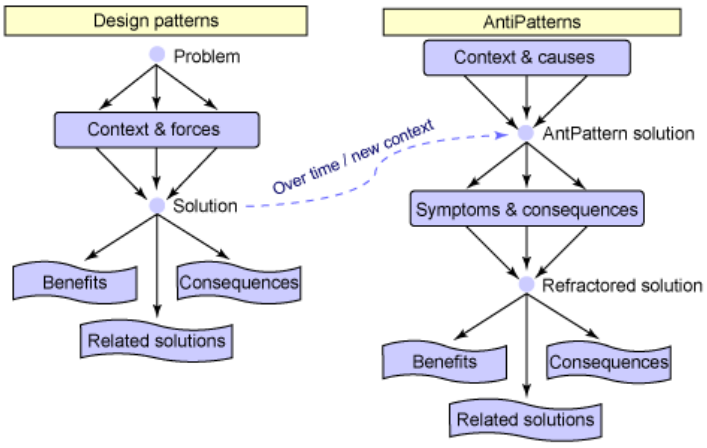

Abbildung 1.11: Patterns und Antipatterns aus [JA05]

Im Gegensatz zu Design-Patterns beschreiben Anti-Patterns oft verwendete, jedoch sehr ineffiziente Lösungen, welche sich für das Projekt als kontraproduktiv herausstellen. Bekannte Beispiele für Anti-Patterns aus der Softwareentwicklung wären Blob, Poltergeist, Spaghetti Code oder Golden Hammer. Ähnlich wie Design-Patterns werden auch Anti-Patterns beschrieben. In [JA05] werden sie nach Problem, Kontext, Symptomen, Konsequenzen, Grundursache und Lösung gegliedert. Es werden Anti-Patterns definiert, welche Fehler bei der Einführung, Umsetzung und Betrieb einer SOA-Landschaft auftreten. Diese können grundlegende Projektentscheidungen betreffen oder Implementierungsdetails. Interessant sind jene Muster, die konkret die betriebliche Einführung einer SOA adressieren sowie grundlegende Designentscheidungen betreffen.

1.9.1 SOA Adoption Anti-Patterns

Hierbei handelt es sich um Muster, die die Einführung von SOA in einem Betrieb verhindern oder verzögern.

Technology Bandwagon

Ein Unternehmen stützt sich auf neue Technologie ohne bedacht abzuwägen welchen Einfluss diese auf ihre Geschäftstätigkeit hat.

- **Problem**
  Die Einführung von SOA wird von der IT und nicht aus der Unternehmung heraus getrieben. Die entstehenden Lösungen sind zwar technisch gut, können die Geschäftsfälle aber nicht hinreichend bedienen.
- **Kontext**
  Dieses Muster tritt meist bei großen Firmen mit einer etablierten IT-Abteilung auf.
- **Symptome**
  Niemand kann die Vorteile der SOA Einführung genau benennen oder die Projekte in denen SOA eingesetzt wird, sind nicht geschäftsrelevant.
- **Konsequenzen**
  Die Kosten für IT steigen ohne merkbar einen ROI (Retun on Investment) zu generieren.
- **Grundursache**
  Der Druck auf Unternehmen sich gegenüber der Konkurrenz als Technologieführer zu etablieren ohne diesen Schritt mit den Bedürfnissen der Unternehmung abzustimmen.
- **Lösung**
  Die Unternehmung darf nicht auf den SOA-Hype aufspringen, sondern erst die zu

lösenden technischen Herausforderungen und Problempunkte der Kundeninteraktion ausfindig machen. Dies führt zu einer Road-Map die hilft die Technologie sauber und mit Bedacht auf die Geschäftsfälle einzuführen.

So, Whats New?

Aus Mangel an Verständnis für den Unterschied zwischen SOA und vorangegangenen Paradigmen stehen die Beteiligten einer Einführung skeptisch gegenüber.

- **Problem**
  Skeptiker im Unternehmen behaupten, SOA ist nur ein neuer Name für alte Techniken und bietet nichts Neues, das nicht bereits umgesetzt wurde.
- **Kontext**
  Tritt auf wenn das IT-Personal sich mit der aktuellen Lösung wohlfühlt oder bereits aufwändige Technologietransformationen mitgemacht hat, welche die Erwartungen nicht erfüllen konnten.
- **Symptome** Das offensichtlichste Merkmal ist der Widerstand von technischen Managern SOA als eine ernsthafte Lösung für Geschäftsprobleme in Betracht zu ziehen.
- **Konsequenzen** Der Widerstand führt dazu, dass geeignete Zeitpunkte einer Einführung verpasst werden.
- **Lösung** Die Unterscheidungspunkte von SOA und früheren Lösungen müssen besser kommuniziert werden. So sollte z.B. der Unterschied zwischen APIs und Diensten besser vermittelt und der Mehrwert von Standards erkannt werden. Ein weiter zu vermittelnder Unterscheidungspunkt ist die Mächtigkeit eines Enterprise-Service-Bus (siehe Kapitel 1.4 „Enterprise Service Bus“) Die beste Lösung jedoch sind erfolgreiche Beispiele, anhand derer man Unterschiede und Erfolgsmerkmale demonstrieren kann.

The Big Bang

Auch bekannt unter dem Namen „Bite more than you can chew.“ Beschreibt den Fall, dass SOA als Allheilmittel angepriesen wird und versucht wird alle Enterprise-Systeme eines Unternehmens auf einen Schlag umzustellen.

- **Problem**
  Dieses Anti-Pattern ist der extreme Gegensatz zu „So, whats new?“. SOA wird als Allheilmittel angepriesen und es wird versucht das gesamte Enterprise-System auf einmal umzustellen.
- **Kontext**
  Ähnelt stark dem Kontext von „Technology Bandwagon“. Personen mit star-

ker technologischer Ausrichtung sind im Unternehmen einflussreicher als ihre geschäftsfokusierten Gegenüber.

- **Symptome**
  Geschäftsabteilungen bringen übermäßig Zweifel an der angestrebeten SOA-Einführung vor.
- **Konsequenzen**
  Dieses Anti-Pattern führt dazu, dass die gemachten Versprechen weit verfehlt werden. Dies kann im schlimmsten Fall dazu führen, dass zu früheren Lösungen zurückgekehrt werden muss. Die Schuld für den Fehlschlag wird SOA zugeschrieben anstatt nach der grundlegenden Ursache zu suchen.
- **Grundursache**
  Zu einflussreiche Personen mit technologischer Ausrichtung können sich über Geschäftsinteressen hinwegsetzen.
- **Lösung**
  Ausgehend von einer Geschäftsstrategie muss SOA die Ziele unterstützen können. Es muss eine Road-Map erstellt werden, die eine schrittweise Einführung, beginnend mit einem Pilotprojekt, vorsieht.

1.9.2 Service Design Anti-Patterns

Diese Kategorie zeigt häufige Fehler auf, die bei der Identifizierung und bei dem Design von Diensten auftreten und zu ineffizienten Lösungen führen.

Webservice = SOA

Werden Webservices mit SOA gleichgesetzt, führt die dazu, dass bei einer Einführung nur existierende APIs durch Webservices ersetzt werden. Das führt zu vielen Diensten die nicht der Geschäftsausrichtung entsprechen.

- **Problem**
  Für viele ist SOA nur ein anderer Name für Webservices. Obwohl die Implementierung von Webservices einen guten Einstiegspunkt in Richtung einer SOA Einführung darstellt, dürfen sie nicht gleichgesetzt werden.
- **Kontext**
  Die meisten der heutigen Webservice Implementierungen entsprechen nicht SOA. Sie sind ein einfacher Ersatz für Remote Procedure Calls oder Punkt-zu-Punkt Nachrichtenübermittlung mittels SOAP.
- **Symptome**
  Existierende APIs werden ohne hinreichende Architektur durch Dienste ersetzt. Neue Dienste entsprechen nicht der Geschäftsausrichtung.

- **Konsequenzen**
  Die Zahl an Diensten nimmt stark zu da jede Schnittstelle als Dienst umgesetzt wird, egal ob sie als SOA-Bestandteil relevant ist oder nicht.
- **Grundursache**
  Mit Hast wird versucht eine SOA-Einführung umzusetzen und dabei wichtige Prinzipien ignoriert.
- **Lösung**
  Es muss ein Plan erstellt werden wie existierende Bestandteile auf SOA übertragen werden sollen und welche Funktionalität als Dienst angeboten werden soll.

The Silo Approach

Dienste werden auf Grund von existierenden Applikationen identifiziert. Das führt dazu, dass Dienste mehrfach und unter anderen Namen entworfen werden.

- **Problem**
  Wenn Dienste basieren auf isolierten Applikationen identifiziert werden anstatt mit Fokus auf die die Geschäftsstrategie.
- **Kontext**
  Unternehmen mit vertikalen funktionalen Modellen, die auf isolierten Applikationen basieren.
- **Symptome**
  Identische Dienste werden von unterschiedlichen Gruppen unter anderen Namen identifiziert.
- **Konsequenzen**
  Da Dienste mehrfach identifiziert und implementiert werden, steigen die Kosten stark an und der Wiederverwendungseffekt einer SOA wird negiert.
- **Grundursache**
  Organisationsgrenzen und Richtlinien verhindern Informationsaustausch und konsolidierte Entscheidungen.
- **Lösung**
  Es müssen Lenkungseinheiten geschaffen werden, die ein koordiniertes Vorgehen über Organisationsgrenzen hinweg ermöglichen.

Misbehaving Registries

Einträge im Dienstverzeichnis sind mehrfach vorhanden. Verzeichnisse überlappen sich und die Verantwortlichkeiten für diese sind nicht hinreichend geklärt. Dies führt zu Verwirrung, schlechter Performance und ungeplanten Mehrkosten.

- **Problem**
  UDDI (Universal Description, Discovery and Integration) Einführung ohne Lenkungsinstanzen führt zu einem ineffizienten Prozess sowie schlechtem Design.
- **Kontext**
  Unternehmen ohne Koordination eines zentralen Verzeichnisdienstes.
- **Symptome**
  Es existieren mehrere, teilweise überlappende Dienstverzeichnisse deren Verantwortlichkeiten nicht klar geregelt sind.
- **Konsequenzen**
  Doppelte Einträge in Verzeichnissen führen zu schwer nachvollziehbarem Laufzeitverhalten und schlechter Performance. Auch können Sicherheitsprobleme auftreten.
- **Grundursache**
  Unklare Verantwortlichkeiten aufgrund des Verfehlens einer unternehmensweiten SOA Strategie.
- **Lösung**
  Kommunikation von Verantwortlichkeiten und Best Practices über alle beteiligten Teams hinweg.

Kapitel 2

SOA und Web Services

In Definition 3, aus Abschnitt 1.2 „Definition von SOA“, wird behauptet, dass SOA mit Web Services implementiert wird. Dies ist durchaus üblich, aber nicht zwingend erforderlich, da Web Services nur eine mögliche Technologie zur Umsetzung darstellen.

> „SOA and web services are two different things, but web services are the preferred standards-based way to realize SOA.“ [Mah05]

Web Services werden in Fachliteratur oft zur Umsetzung von SOA empfohlen. Der Vorteil von Web Service ist die auf Standards basierende Interoperabilität, welche den Kollaborateuren - in SOA den Diensten und ihren Konsumenten - es erst ermöglicht eine „gemeinsame Sprache“ zu sprechen.

> **Definition: Web Services**
> „...software applications 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 softwareapplications using XML based messages via Internet based protocols .“ [W3C02]

Da alle involvierten Dokumente bei Web Services auf XML basieren bleibt die Plattformunabhängigkeit garantiert. Die Grundbausteine von WebServices

- **SOAP** - Simple Object Access Protocol
- **WSDL** - Web Service Description Language
- **UDDI** - Universal Description, Discovery and Integration

auf welche im Folgenden noch ausführlich eingegangen wird, integrieren sich gut in das SOA Konzept.

> „Der mit Abstand vielversprechendste Ansatz (zur Realisierung von SOA) sind derzeit Web Services. Neben den theoretischen Grundlagen gibt es auch schon sehr weitgehende Spezifikationen und Implementierungen. Dadurch können die meisten Konzepte auf Basis von Web Services getestet, veranschaulicht und umgesetzt werden“ [Mel10]

2.1 SOAP - Simple Object Access Protokol

SOAP bedeutete ursprünglich Simple Object Access Protokoll, jedoch ist diese Bezeichnung kaum mehr gebräuchlich, da sie für die Verwendung nicht zutreffend ist und SOAP wird meist als Eigenname benutzt. Es ist ein XML basiertes Protokoll und baut zur Übertragung von Daten auf gängige Protokolle der Anwendungs- und Transportschicht. Die gebräuchlichste Kombination ist hierbei HTTP/TCP.
SOAP überträgt konkrete Instanzen von Nachrichten, die in einem WSDL-Dokument definiert sind.

Listing 2.1: Listing 2: SOAP Message

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version=\"1.0\"?>
<soap:Envelope
xmlns:soap=\"http://www.w3.org/2001/12/soap-envelope\"
soap:encodingStyle=\"http://www.w3.org/2001/12/soap-encoding\">

<soap:Body xmlns:m=\"http://www.example.org/stock\">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

Listing 2 zeigt einen einfache SOA-Request Nachricht. Sie besteht aus Header, Envelope und Body in dem die Operation und ihre Parameter codiert sind.

2.2 WSDL - Web Services Description Language

Ein WSDL Dokument beschreibt Dienste als eine Menge von Netzwerk-Endpoints. Ein Endpoint wird definiert durch eine Netzwerkadresse und einem wiederverwendbaren Binding. Eine Menge von Endpoints stellen einen Service dar. Ein WSDL Dokument benutzt folgende Elemente zur Definition eines Services:

- **Types**
  Ein Kontainer für Datentypdefinitionen. (Am gebräuchlichsten ist hier XSD)
- **Messages**
  Eine abstrakte, typbehaftete Definition der ausgetauschten Daten
- **Operation**
  Eine abstrakte, typbehaftete Defintion der Aktionen, die ein Service unterstützt
- **Port Type**
  Eine abstrakte Menge von Operationen, die von einem oder mehreren Endpoints unterstützt werden.
- **Binding**
  Ein konkretes Protokoll und Datenformat, das für einen Endpoint spezifiziert wird.

- **Port**
  Ein Endpunkt, der sich aus eienr Kombination aos Binding und Netzwerkadresse zusammensetzt.
- **Service**
  Eine Menge zusammengehöriger Endpoints.

Als Beispiel für das Zusammenspiel von WSDL und XSD wird hier die Definition einer Serviceoperation, ihrer Nachrichten und Datentypen genauer vorgestellt. Da WSDL Dokumente durch ihren XML Aufbau eine enorme länge bekommen können werden hier nur die für das Beispiel relevanten Ausschnitte aus den jeweiligen Dokumenten übernommen. Abbildung 2.1 zeigt, woher diese Ausschnitte kommen.

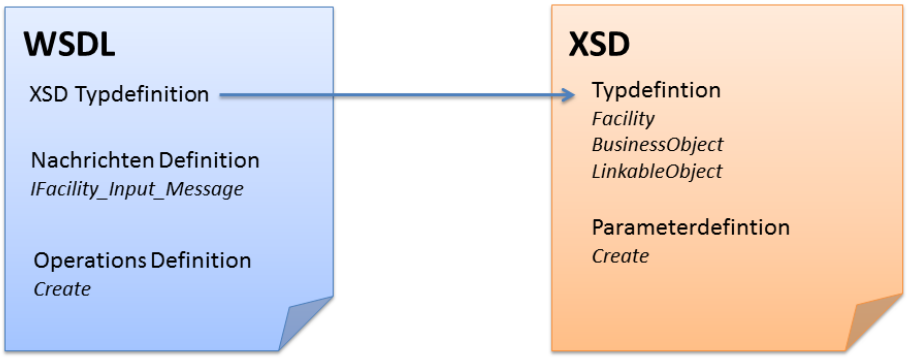

Abbildung 2.1: Ursprung der WSDL/XSD Ausschnitte

Listing 3 zeigt eine XSD Datentypdefinition, die an ein WSDL-Dokument angehängt ist. Wie man sieht können auch abgeleitete Typen beschrieben werden, was bedeutet, dass Objekte abgeleiteter Klassen korrekt über SOAP serialisiert und deserialisiert werden können. Das konkrete Beispiel aus der CLM-Plattform zeigt den Typ „Facility“, welcher vom Typ „Bussiness Object“ und dieser wiederum von „Linkable Object“ abgeleitet ist.

Listing 2.2: Listing 3: XSD Typdefintion

```
<xs:complexType name="Facility">
  <xs:complexContent mixed="false">
    <xs:extension base="tns:BusinessObject">
      <xs:sequence>
        <xs:element name="Name" nillable="true" type="xs:string" />
        <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string" />
        <xs:element minOccurs="0" name="Address" nillable="true" type="xs:string" />
        <xs:element minOccurs="0" name="CountryRegion" nillable="true" type="xs:string" />
        <xs:element minOccurs="0" name="PostalCode" nillable="true" type="xs:string" />
        <xs:element minOccurs="0" name="PostalTown" nillable="true" type="xs:string" />
        <xs:element minOccurs="0" name="Geocode" nillable="true" type="xs:string" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="Facility" nillable="true" type="tns:Facility" />
<xs:complexType name="BusinessObject">
  <xs:complexContent mixed="false">
    <xs:extension base="tns:LinkableObject">
      <xs:sequence />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="BusinessObject" nillable="true" type="tns:BusinessObject" />
<xs:complexType name="LinkableObject">
  <xs:sequence>
```

```
27      <xs:element name="Id" type="xs:long" />
28    </xs:sequence>
29  </xs:complexType>
30  <xs:element name="LinkableObject" nillable="true" type="tns:LinkableObject" />
31  <xs:complexType name="Customer">
32    <xs:complexContent mixed="false">
33      <xs:extension base="tns:BusinessObject">
34        <xs:sequence>
35          <xs:element name="Name" nillable="true" type="xs:string" />
36          <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string" />
37        </xs:sequence>
38      </xs:extension>
39    </xs:complexContent>
40  </xs:complexType>
```

Listing 4 zeigt eine Operationsdefinition in einem WSDL-Dokument. Hier wird die Operation „Create“ des Anlagendienstes beschrieben. Er dient dazu eine neue Anlage anzulegen und erwartet hierzu eine „IFacilityCreateInputMessage“ und definiert die Rückgabe einer „IFacilityOutputMessage“. Diese Nachrichten sehen wie die in Listing 2 aus nur, dass der erwartete Parameter dem in Listing 3 gezeigten komplexen Type „Facility“ entspricht.

Listing 2.3: Listing 4: WSDL Operation-Definition für Create

```
1  <wsdl:portType name="IFacilityService">
2  <wsdl:operation name="Create">
3  <wsdl:input wsaw:Action="http://ntesystems.at/clm/v2/IFacilityService/Create"
       message="tns:IFacilityService_Create_InputMessage" />
4  <wsdl:output wsaw:Action="http://ntesystems.at/clm/v2/IFacilityService/CreateResponse"
       message="tns:IFacilityService_Create_OutputMessage" />
5  <wsdl:fault wsaw:Action="http://ntesystems.at/clm/v2/IFacilityService/CreateScSecurityFaultFault"
       name="ScSecurityFaultFault"
       message="tns:IFacilityService_Create_ScSecurityFaultFault_FaultMessage" />
6  <wsdl:fault
       wsaw:Action="http://ntesystems.at/clm/v2/IFacilityService/CreateScInvalidArgumentFaultFault"
       name="ScInvalidArgumentFaultFault"
       message="tns:IFacilityService_Create_ScInvalidArgumentFaultFault_FaultMessage" />
7  <wsdl:fault wsaw:Action="http://ntesystems.at/clm/v2/IFacilityService/CreateScFaultFault"
       name="ScFaultFault" message="tns:IFacilityService_Create_ScFaultFault_FaultMessage" />
8  <wsdl:fault
       wsaw:Action="http://ntesystems.at/clm/v2/IFacilityService/CreateScDataManipulationFaultFault"
       name="ScDataManipulationFaultFault"
       message="tns:IFacilityService_Create_ScDataManipulationFaultFault_FaultMessage" />
9  </wsdl:operation>
```

Listing 5 zeigt die Definition der Nachricht im WSDL-Dokument, welche wiederum auf das XSD mit der Typendefinition der Create-Parameters verweist. Dort wird ersichtlich, dass als Parameter ein Facility-Object gefordert wird und ein Customer-Object, welchem dieses zugeordnet werden soll.

Listing 2.4: Listing 5: Input Message Definition aus WSDL

```
1  <wsdl:message name="IFacilityService_Create_InputMessage">
2      <wsdl:part name="parameters" element="tns:Create" />
3  </wsdl:message>
```

Listing 2.5: Listing 6: XSD Definition der Parameter der Input-Message

```
1  <xs:element name="Create">
2      <xs:complexType>
3          <xs:sequence>
4              <xs:element minOccurs="0" name="newFacility" nillable="true" type="q1:Facility"
                   xmlns:q1="http://ntesystems.at/clm/v2" />
5              <xs:element minOccurs="0" name="owner" nillable="true" type="q2:Customer"
                   xmlns:q2="http://ntesystems.at/clm/v2" />
6   </xs:sequence>
7  </xs:complexType>
```

Dieses Beispiel zeigt die Kernelemente eines WSDL-Dokuments. Die Verkettungen sind sehr verschachtelt und bei weitem nicht einfach für Menschen zu interpretieren. Wie aber schon im Kapitel 1.2.1 „Dienst“ erwähnt sollen Service-Beschreibungen Maschinen-lesbar sein um daraus seine konkrete Verwendung ableiten zu können.

2.3 UDDI - Universal Description, Discovery and Integration

> „UDDI is dead. As in, you know, not pinin', passed on, is no more, ceased to be, expired and gone to meet 'is maker.“ *(Stefan Tilkov, InnoQ - SOA Consulting)*

UDDI (Universal Description, Discovery and Integration) ist eine Umsetzung des in Kapitel 1.2.3 „Dienstverzeichnis“ vorgestellten Konzepts des Dienstverzeichnisses. Es soll die Möglichkeit bieten Dienste zu registrieren und zur Laufzeit aufzufinden. Die Idee war es ein öffentliches Dienstverzeichnis (analog zu einem Telefonbuch) zu schaffen in das unterschiedlichste Anbieter ihre Dienste registrieren können und in diesem Dienstnutzer die von ihnen benötigte Funktionalität finden können.

> „The UDDI Business Registry (UBR) was part of the UDDI Project announced in September 2000. The project goals were to define a set of specifications to enable description, discovery and integration and to prove interoperability through a shared implementation of those specifications and provide feedback to refine the specifications through operational experience. [...] The primary goal of the UBR was to prove the interoperability and robustness of the UDDI specifications through a public implementation. This goal was met and far exceeded. The UBR ran for 5 years, demonstrating live, industrial strength UDDI implementations managing over 50,000 replicated entries.“ [Mic05a]

Die Entwicklung wurde von IBM, Microsoft und SAP getrieben, sind jedoch 2005 wieder eingestellt worden. Obwohl die Firmen behaupteten, das öffentliche Dienstverzeichnis war ein Erfolg und beweist die Funktionalität von UDDI, gilt dieses in der SOA-Comunity als fataler Fehlschlag. Grund für das Scheitern war, dass das Verzeichnis nach den fünf Jahren, in denen es bestanden hat, mit unnützen und nichtmehr funktionalen Diensten zugemüllt war und für Nutzer es kaum möglich war aus der Fülle brauchbare Dienste ausfindig zu machen.

2.4 WS-* Spezifikationen

Bei den WS-* Spezifikationen handelt es sich um eine Vielzahl an unabhängigen Standards, die jeder für sich den Einsatz von Webservices mit SOAP/WSDL um ein spezielles Anwendungsgebiet erweitern. Das heißt, sie können unabhängig voneinander eingesetzt und beliebig kombiniert werden. Sie erweitern die Basisfunktionalität von SOAP/WSDL.

Forrester Reasearch zählte bereits 2007 ca. 115 Standards rund um SOAP und Webservices. Ein Großteil davon machen die WS-* Standards aus, jedoch finden nur einige breite Verwendung. Grund dafür ist, dass die schiere Zahl für Verunsicherung in der

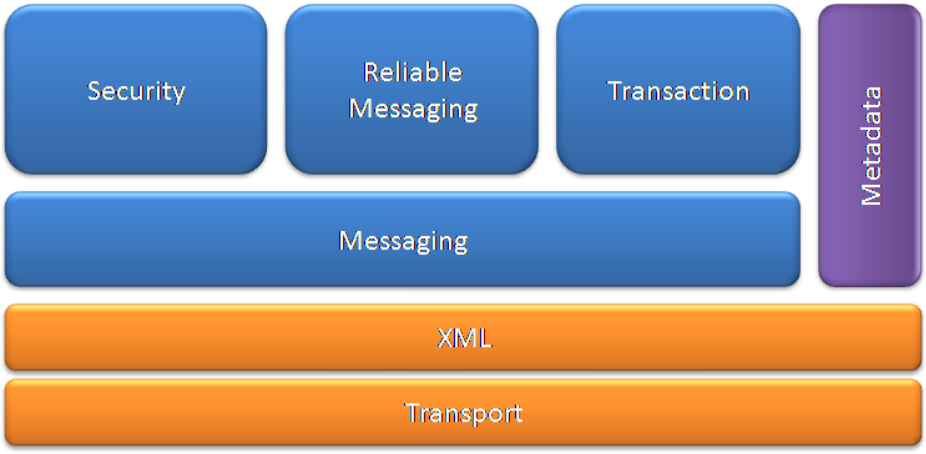

Abbildung 2.2: Überblick über die Teilbereiche der WS-* Spezifikationen

Webservice-Community sorgt, da jede Spezifikation für sich den Zyklus von Definition, Ratifizierung und Update durchläuft und nicht klar ist, inwieweit Interoperabilität in Zukunft gewährleistet sein wird. Darum setzen die meisten Implementierungen nur ein Basisset dieser Spezifikationen um.
Die WS-* Spezifikationen sind notwendig, da SOAP den Inhalt seiner Header und Body-Elemente nicht definiert und um interoperabel zu bleiben es Standards bedarf. Sie entstehen also aus dem Bedürfnis heraus allgemeine Anforderungen an Nachrichten festzulegen. Zum Beispiel wie einheitlich Benutzer-Credentials in SOAP-Headers ausgetauscht werden. Um nicht kompatible Eigenentwicklungen zu vermeiden wurde etwa der WS-Security Standard geschaffen.
Aufgrund der Anzahl an Spezifikationen können hier nur die Pakete, die mehrere Spezifikationen unter sich vereinen, kurz angerissen werden. Einen guten Überblick über die WS-* Standards findet man unter [Gmb07].

- **Transport**
  Um interoperabel zu bleiben werden Webservices meist über HTTP oder HTTPS laufen. Die WS-Spezifikationen im Transport-Paket beschreiben, wie Nachrichten über andere Protokolle übertragen werden und wie Adressinformationen in den SOAP-Header codiert werden.

- **XML**
  Am meisten hervorzuheben aus diesem Paket sind die Spezifikationen WS-Digital Signature und WS-Encryption zur Sicherung von XML-Dokumenten und SOAP-Nachrichten, die der WS-Security Spezifikation zugrunde liegen.

- **Messaging**
  Dieses Paket beinhaltet die WS-Addressing Spezifikation, welche Adressierung und Routing von der Transportschicht entkoppelt und es erlaubt auf standardisiertem Weg SOAP-Nachrichten protokollunabhängig über mehrere Hops zu versenden. Die ebenfall enthaltene WS-Transfer Spezifikation beschreibt, wie CRUD-Operationen auf Ressourcen mit SOAP-Nachrichten ausgeführt werden

und ermöglicht so die Implementierung von REST-Services [1].

- **Security**
  WS-Security ist eine der frühesten WS-* Spezifikationen . Ursprünglich aus dem Bedarf geschaffen zu standardisieren, wie Credentials in SOAP-Nachrichten serialisiert werden, hat sich der Standard stark erweitert und definiert z.B. den Aufbau von Trust-Relationships, Austausch von Security Tokens und Umsetzungen von Federated-Trust Konzepten.

- **Reliable Messaging**
  Die WS-* Spezifikationen in diesem Gebiet beschreiben, wie erreicht werden kann, dass:

  - Nachrichten verlässlich, nur einmal und geordnet ankommen.
  - Diese Eigenschaft protokollunabhängig ist
  - Dies über mehrere Hops sichergestellt ist.
  - Für eine komplette SOAP-Nachricht gilt und nicht etwa nur für ein IP-Paket.

  Hier konkurrieren zwei WS-* Spezifikationen, WS-Reliability und WS-Reliable Messaging, untereinander.

- **Transaction**
  WS-Coordination, WS-Atomic Transaction und WS-Business Activity sind Protokolle, die es erlauben Transaktionen über Plattformgrenzen hinweg aufzuspannen. Wobei WS-Coordination spezifiziert, wie Teilnehmer in einem Coordination-Context zu registrieren sind und wie dieser über Plattfromgrenzen hinweg weitergegeben wird. WS-Business Activity stellt eine Implementierung dieses Protokolls dar und WS-Atomic Transaction, wie ein two-phase-commit (2PC) auf diesen Kontext auszuführen ist.

[1] REST steht für Representational State Transfer und stellt eine weitere, ressourcenbasierte Möglichkeit zur Entwicklung von Webservices dar welche erstmals von Roy Fielding in [RTF02] vorgestellt wurde.

Kapitel 3

Umsetzung von SOA für NTE.CLM

3.1 Projektüberblick

3.1.1 Control Loop and Monitoring Plattform

NTE.CLM steht für NTE „Control Loop and Monitoring“ und stellt eine Plattform dar, über welche eine Vielzahl von unterschiedlichen Anlagen überwacht und manipuliert werden können.

> **Definition 1: Anlage** Unter Anlage wird in diesem Kontext ein Gebäude verstanden, in welchem technisches Equipment zu unterschiedlichen Zwecken verbaut ist. Über Sensoren kann der Zustand dieser Anlage gelesen und über Aktoren manipuliert werden. Ein Beispiel hierfür wäre eine Wohnanlage mit Fernwärmeanschluss und Solaranlage.

Die Basisfunktionalität der CLM-Plattform entspricht hier der von SCADA-Systemen (Supervisory Control and Data Acquisition), also das Überwachen, Visualisieren, Steuern und Regeln von Anlagen.

Jedoch gibt es auch Abgrenzungskriterien. Die Plattform ersetzt keine bestehenden Regelungen sondern versteht sich als Bindeglied um Daten aus unterschiedlichen Quellen zusammenzuführen, deren Status zu visualisieren, die Möglichkeit zu geben Ferndiagnosen durchzuführen und gegebenenfalls von entfernter Stelle aus Aktionen setzen zu können. Hauptaugenmerk liegt hierbei auf der Abstrahierung der darunterliegenden Technologien (Bussysteme, SPS, Steuereinheit), der Aufzeichnung der Historie einer Anlage, der Aufbereitung der Daten sowie Fernsteuerung und Ferndiagnose über das Internet. Im Gegensatz zu SCADA-Systemen regelt die Plattform keine Produktionsprozesse, sondern konzentriert sich auf Abläufe, die den Energiehaushalt einer Anlage regeln.

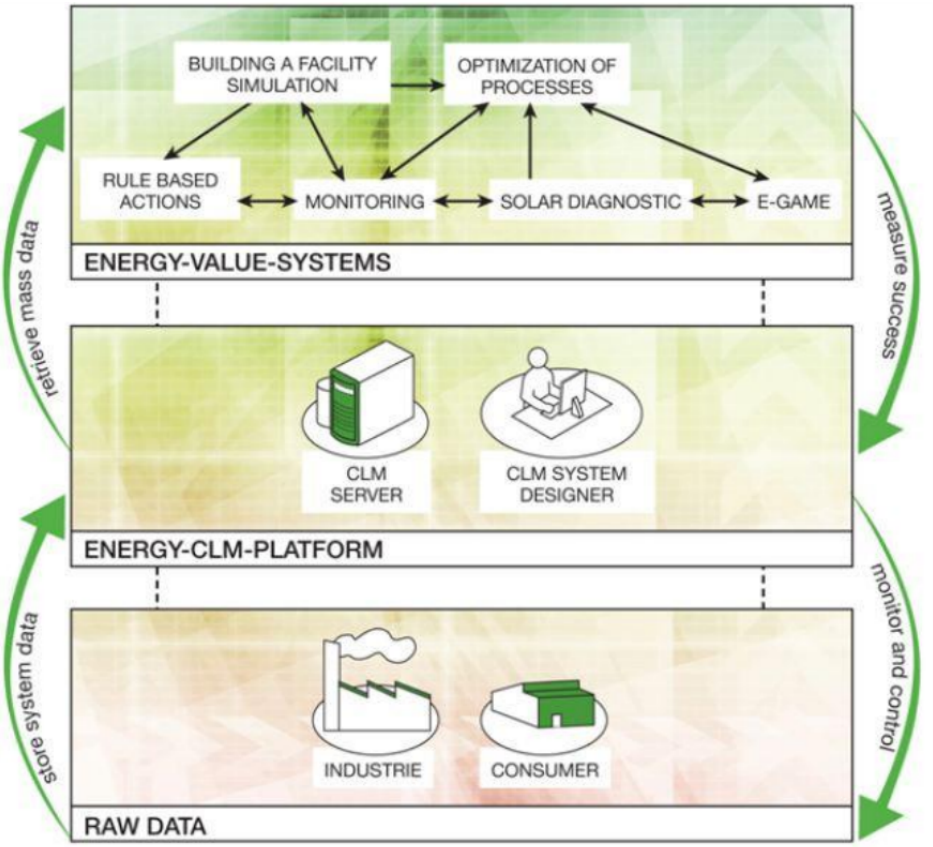

Abbildung 3.1: Überblick der Aufgaben der NTE.CLM - Plattform

3.1.2 Energy Value Systems

Aufbauend auf der Basisfunktionalität stellt die Plattform ihren Benutzern sogenannte „Energy-Value-Systems“ zur Verfügung. Dabei handelt es sich um Funktionalität welche es dem Benutzer ermöglicht Informationen über den Energieverbrauch und Optimierungspotenzial seiner Anlagen zu erlangen und gegebenenfalls sofort Aktionen zu setzen. Beispiele für „Energy-Value-Systems“ sind:

- **Energiebuchhaltung**
  Aus den erfassten Basisdaten werden Energiekennzahlen von Anlagen errechnet und in Echtzeit[1] visualisiert. Des Weiteren sollen detaillierte Berichte über den Energiehaushalt von Anlagen erstellt werden können, die als Energieausweis für diese Anlage dienen. Unter Energiekennzahlen versteht man domänenspezifische Kenngrößen, die sich auf den Energieverbrauch beziehen.

  Beispiel
  Allgemein: kWh / (Tag, Jahr, Monat)
  Einzelhandel: kWh / m^2 Verkaufsfläche
  Gastronomie: kWh / Gastplatz

[1]Die Definition von Echtzeit wird in Kapitel 3.6 „SOA Performance im Monitoring gegeben.

Diese Kennzahlen machen unterschiedliche Standorte vergleichbar.

- **Regelbasierte Aktionen**
  Hierbei handelt es sich um ein Optimierungssystem (rule based optimization), das anhand von bestimmten Regeln Optimierungspotenziale aufzeigt.

  Beispiel
  Wenn die Fenster geöffnet sind, dann ist die Heizung abzuschalten.

  Es wird permanent von der Monitoring-Einheit mit Ereignissen versorgt um die Aktualität zu gewährleisten.

- **Diagnostik**
  Spezialisiert sich momentan nur auf Solardiagnose und diagnostiziert permanent die Funktionsweise der Solaranlagen. Durch eine Wissensdatenbank sollen anhand von markanten Messwertveränderungen Ursachen für Fehlfunktionen automatisch gefunden werden können.

  Beispiel
  Stagnation, Defekter Temperatursensor, Schnee auf Solarfläche.

  Ziel des Diagnose-Verfahrens ist es, ohne zusätzlichen Messaufwand, die Funktionstüchtigkeit von Solarthermieanlagen zeitnahe zu überwachen, deren erwarteten Ertrag zu überprüfen und die Ursache von auftretenden Fehlern gezielt dem Benutzer mitzuteilen um eine zielgerichtete Instandsetzung zu ermöglichen. Um diese Zielvorgabe erreichen zu können, ist eine Kombination aus diversen Fehlerdetektionsverfahren notwendig, wobei verschiedene Auffälligkeiten über logische Bedingungen und Wahrscheinlichkeitsverteilungen zu konkreten Fehlerursachen verknüpft werden. Dadurch soll es möglich sein multiple Anlagenfehler auf eine oder mehrere Fehlerursachen zurückzuführen. Ohne effiziente und breite Datenbasis sind hier zwar die theoretischen Grundlagen umsetzbar, diese jedoch nicht in adäquater Zeit zu einer verifizierten Qualität zu führen. Im Rahmen mehrerer Iterationen wird hier eine laufende Verbesserung nötig sein.

- **Gebäudesimulation**
  Anhand der erhobenen Daten soll eine dynamische Gebäudesimulationen durchgeführt werden. Auf Basis der durch die Gebäudesimulation erhaltenen „idealen“ Energiebedarfswerte werden Funktionen erarbeitet, die das entsprechende Gebäudeverhalten in ausreichender Genauigkeit wiedergeben. In weiterer Folge sollen die aktuellen Wetterdaten, die im Zuge des Monitorings erfasst werden, als Eingangsgrößen der generierten Funktionen fungieren und damit den idealen Ist-Energiebedarf des Versorgungsobjektes errechnen. Ziel ist die Generierung einer Funktion, die das Gebäudeverhalten in ausreichender Näherung widerspiegelt, welche den Vergleichswert für den aktuellen Ist-Verbrauch des Versorgungsobjektes liefert. Inwieweit die Erstellung von Energiemodellen in diesem Bereich automatisiert werden kann, steht zum Zeitpunkt der Erstellung dieser Arbeit noch nicht fest.

- **E-Game**
  Konkrete Umsetzungen von E-Games stehen noch aus. Die Grundintention besteht aber in der spielerischen Bewusstseinsbildung über das Optimierungspotenzial des Energiehaushalts. Es soll Anreize für den Nutzer schaffen, Energieoptimales Verhalten zu trainieren und so ihrerseits einen Teil zum Optimierungsprozess beitragen zu können.

Durch die enge inhaltliche Kopplung der Systeme wird gewährleistet, dass die Systeme, die auf eine breite Datenbasis angewiesen sind diese in der CLM-Plattform teilen können, um ihre volle Effizienz zu entfalten. Programmtechnisch sollen diese Systeme jedoch entkoppelt funktionieren und die Möglichkeit zur Erweiterung um neue „Energy-Value-Systeme“ jederzeit offen bleiben.

Diese Arbeit befasst sich mit dem Design und der Implementierung der Service-Infrastruktur und deren Backends, aber nicht mit der Entwicklung der einzelnen „Energy-Value-Systeme“.

3.2 Anforderungen

Um die Anforderungen des Projekts erfassen zu können wurde ein Rollenkonzept entwickelt, welches die Akteure im System identifiziert. Für diese Rollen können unterschiedliche Anforderungen erhoben werden.

- **Managed Service Provider** Ist im Besitz einer Software Plattform, mit der verteilte Anlagen verwaltet werden können.
- **Service Hoster**
  Ist verantwortlich für die Server-seitige Hardware (Serverfarm). Er stellt diese gegen Entgelt zur Verfügung und garantiert deren Verfügbarkeit.
- **Service Provider**
  Typischerweise eine Firma, deren Dienstleistung darin besteht, ihren Kunden vor Ort eine Anlage zu installieren und zu betreuen.
- **Service Manager**
  Ist Angestellter des Service Providers und verantwortlich für die Betreuung der Kunden bzw. deren Anlagen.
- **Service Installer**
  Ist Angestellter des Service Providers und verantwortlich für die Installation und Betreuung der Anlage vor Ort. Er ist typischerweise dem Service Manager unterstellt.
- **Service User**
  Er besitzt und/oder benützt die Anlage die vom Service Provider installiert und betreut wird.

Weiters wurden die Rollen mit unterschiedlichen Ausprägungen von fiktiven Personen und Firmen besetzt um Anforderungen zu verfeinern und Geschäfts-Szenarien aufstellen und evaluieren zu können. Diese sind jedoch für diese Arbeit von geringerer Relevanz und können aus Vollständigkeitsgründen hier nicht diskutiert werden.

3.2.1 Anforderungen der Systemrollen

Aus dem in diesem Projekt eingesetzten Softwareentwicklungsprozess heraus (SCRUM), in welchem die Anforderungen später in „User Storys“ abgebildet werden, sind auch hier schon die Anforderungen der noch abstrakten Systemrollen, gleich wie „User Storys“ in der Ich-Form gehalten. Diese Auflistung enthält die erhobenen Anforderungen und sie entsprechen einer „Wunschliste“ an das System. Die Anforderungen einzelner Systemrollen können im Widerspruch stehen oder technisch kaum realisierbar sein, geben jedoch die Zielrichtung für das System vor.

3.2.2 Auszüge aus den Anforderungen

Insgesamt wurden ca. 550 Anforderungspunkte erhoben. Diese stellen, wie schon oben erwähnt, natürlich nicht die Gesamtfunktionalität des Systems dar sondern sollen als Vorgabe dienen, in welche Richtung sich das Produkt entwickeln soll. Diese werden vom Product Owner[2] erstellt, der diese aus seiner persönlichen Markterfahrung und aus Gesprächen mit potenziellen Kunden und Nutzern sowie Fachleuten ableitet.

Hier wird nur ein Auszug aus den Anforderungen des Managed Service Providers und des Service Users vorgestellt. Allgemein ist zu betrachten, dass sich Anforderungen, die leichter auf die Architekturauswahl umzulegen sind, eher am oberen Ende der Systemrollen befinden. Der Service User hat natürlich mehr Anforderungen an konkrete Prodktfunktionalität.

3.2.3 Managed Service Provider

Skalierung

- Ich möchte mit dem System beliebig viele Kunden bedienen können.
- Ich möchte, dass die Antwortzeit des Systems unabhängig von der Anzahl der Kunden ist.
- Ich möchte, dass die Qualität des Services unabhängig von der geographischen Lage meiner Kunden ist.
- Ich möchte, dass die Funktionalität des Systems jederzeit erweitert werden kann.
- Ich möchte, dass die Hardware des Systems jederzeit erweitert werden kann.
- Ich möchte, dass die Software möglichst unabhängig von der Hardware ist.
- Ich möchte, dass das System in einer stark verteilten Umgebung läuft.
- Ich möchte, dass das System auf einem einzelnen Rechner läuft.

Anschaffungskosten

- Die benötigte (Dritt-)Software soll möglichst günstig sein.
- Die benötigte Hardware soll möglichst günstig sein.
- Ich möchte meinen Kunden verschiedene Varianten meines Systems anbieten können, um Ihnen ein optimales Preis/Leistung Verhältnis zu bieten.

Betriebskosten

- Ich möchte, dass Software Updates/Patches keine Kosten verursachen
- Ich möchte die Energie Kosten für die Server Farm gering halten.

Wartungsaufwand

[2]Hierbei handelt es sich um einen Begriff des Entwicklungsprozesses (SCRUM): Der Product Owner legt das gemeinsame Ziel fest, das das Team zusammen mit ihm erreichen muss. Zur Definition der Ziele dienen ihm meist User Stories, wobei auch andere Techniken erlaubt sind. Er setzt regelmäßig die Prioritäten der einzelnen Product-Backlog-Elemente Dadurch legt er fest, welches die wichtigsten Features sind, aus denen das Entwicklungsteam eine Auswahl für den nächsten Sprint trifft. [Wik10d]

- Ich möchte, dass dafür möglichst wenig Personal benötigt wird.
- Ich möchte, dass das System möglichst einfach Konfiguriert werden kann.
- Ich möchte, dass auftretende Fehler schnell erkannt werden.
- Ich möchte, dass auftretende Fehler zurückverfolgt werden können.
- Ich möchte, dass auftretende Fehler schnell behoben werden können.

Sicherheit

- Ich möchte eine sichere Kommunikation.
- Ich möchte eine sichere Datenhaltung.
- Ich möchte, dass die Daten vertraulich behandelt werden.

Verfügbarkeit

- Ich möchte meinen Kunden einen 24x7 Betrieb des Systems ermöglichen.
- Ich möchte eine meinen Kunden eine bestimmte Verfügbarkeit (Uptime) garantieren können.

Effizienz

- Ich möchte mit wenig Aufwand möglichst viele Kunden bedienen können.
- Ich möchte auf einem System mehrere Kunden betreuen können.

Anpassung

- Ich möchte, dass sich das System und insbesondere die graphischen Benutzeroberfläche vom Kunden entsprechend seinen Anforderungen anpassen lassen.
- Die Sprache soll auswählbar sein.
- Lokale kulturelle Eigenheiten sollen konfiguriert werden können.

3.2.4 Service User

Visualisierung

- Ich möchte eine visuelle Darstellung vom Aufbau der Anlage haben, die auch für Laien von Nutzen ist.
- Ich möchte eine visuelle Darstellung der Funktionsweise der Anlage haben, die auch für Laien von Nutzen ist.

Effizienz der Anlage

- Ich möchte sehen, wie viel Energie die Anlage produziert.
- Ich möchte sehen, welchem Geldwert die produzierte Energie entspricht.
- Ich möchte erkennen, ob sich die Anlage schon amortisiert hat.
- Ich möchte erkennen, ob die Anlage optimal eingestellt ist.
- Ich möchte erkennen, wie effizient meine Anlage im Vergleich zu anderen ist.
- Ich möchte erkennen, ob meine Anlage gerade mehr Energie produziert als eigentlich verbraucht wird.

- Ich möchte, dass das System auf Basis aktueller Werte Empfehlungen abgibt, gewisse Ressourcen (z.B. Warmwasser) momentan eher zu nützen oder nicht.
- Ich möchte, dass das System auf Basis von Wetterdiensten Prognosen abgibt, wie günstig gewisse Ressourcen (z.B. Warmwasser) in naher Zukunft sein werden.

Konfigurieren der Anlage

- Ich möchte bestimmte Werte der Anlage konfigurieren (schreiben) können.

Reporting

- Ich möchte mir den Verlauf von Werten der Anlage anzeigen lassen.
- Ich möchte den Verlauf verschiedener Werte miteinander vergleichen.
- Ich möchte, dass mir das System einen Bericht über einen bestimmten Zeitraum erstellt.
- Ich möchte bestimmte Daten der Anlage ausdrucken können.

Verfügbarkeit

- Ich möchte im Haus Zugang zum System haben
- Ich möchte von unterwegs Zugang zum System haben.

Erweiterbarkeit/Integration

- Ich möchte, dass sich das System erweitern/integrieren lässt.
- Ich möchte mit dem System nicht nur meine Solaranlage verwalten, sondern auch andere Ressourcen (Fernwärme, Zentralheizung, Strom, etc.) verwalten.

3.2.5 Anforderungen an die Architektur

Die Anforderungen aus den System-Rollen beinhalten sowohl Anforderungen an einzelne Energy-Value-Systeme als auch Qualitätsanforderungen an die Plattform an sich. Diese wurden identifiziert als:

Verteiltheit
Aus der Natur der Dienstleistung heraus ist es notwendig, dass alle Komponenten in einer verteilten Umgebung miteinander operieren können. Steuereinheiten sind in den Anlagen verbaut. Ihre aktuellen und historischen Daten müssen ortsunabhängig über das Internet visualisiert und verarbeitet werden. Möglicherweise auch, wenn momentan keine Verbindung zur Anlage besteht.

Interoperabilität Verschiedene „Energy-Value-Systeme“ sollen mit der Plattform interagieren können.

Skalierbarkeit Das System muss mit der Anzahl der Nutzer skalieren. Das System muss aber auch nach Anzahl der teilnehmenden „Energy-Value-Systemen“ skalieren.

Sicherheit Die erhobenen Daten sind von juristischer Relevanz und müssen daher geschützt werden. Das bedeutet das alle Kommunikationskanäle sicher sein müssen und das System für den Zugriff hinreichende Autorisierungs- und Authentifizierungsmechanismen bietet. Auch muss der Zugriff auf einzelne „Energy-Value-Systeme“ für verschiedene Benutzer einstellbar sein.

Performanz Es müssen große Datenmengen verarbeitet werden. Pro Sensor kann es im Sekundentakt Daten geben. Bis zu 3000 Messpunkte können pro Anlage auftreten. Für Auswertungen auf historische Daten bedeutet das eine hohe Datenmenge. Die Benutzer erwarten sich angemessenes Antwortverhalten.

„Live“-Daten müssen für den Benutzer in Echtzeit [3] verfügbar sein. Dies gilt auch für den Zugriff über das Internet, wo es sehr schwierig ist Antwortzeiten zu garantieren.

Erweiterbarkeit Neue „Energy-Value-Systeme“ müssen leicht an das bestehende System angebunden werden können. Diese Systeme können auch von Drittherstellern kommen.

Verfügbarkeit Das System muss eine relativ hohe Verfügbarkeit aufweisen da Daten nicht verloren gehen dürfen und für Benutzer jederzeit erreichbar sein sollen.

Nachverfolgbarkeit Aktionen des Systems müssen nachverfolgbar sein. Aus Rechtsgründen muss ermittelbar sein welche Aktion von System oder einem Benutzer ausgelöst wurde und wie der Status einer Anlage zu dieser Zeit war. Systemfehler müssen erkannt und gegebenenfalls an einen Fehlerarbeitsplatz weitergeleitet werden können.

3.2.6 Erwartungen an SOA

Die Entscheidungsträger erwarten sich von einer serviceorientierten Architektur die Interoperabilität zwischen „Energy-Value-Systemen“ und der CLM-Plattform in Zukunft gewährleisten zu können, da das System je nach Geschäftsfall skalieren kann und leicht um neue „Energy-Value-Systeme“ erweitert werden kann.

Vergleicht man diese Forderungen und die aus den System-Rollen mit den Qualitätsattributen von SOA (siehe Kapitel 1.7 „Qualitätsattribute“) kann man technologisch noch zu lösenden Problembereiche erkennen. Wählt man WebServices als Technologie zur Umsetzung von SOA, verbessert sich der Sicherheitsaspekt, da der WebService-Standard Bindings für verschlüsselte Kanäle zur Verfügung stellt. Da für die Entwickler zur Zeit des Projektstarts noch nicht feststeht, in welche Richtung sich die Plattform entwickeln wird, da dies hauptsächlich von den Anforderungen der ersten

[3] Als Echtzeit wurden für diese Anwendungen fünf Sekunden festgesetzt.

Forderungen	SOA	SOA mit Webservices
Verteiltheit	-	gut
Interoperabilität	gut	gut
Erweiterbarkeit	gut	gut
Skalierbarkeit	neutral	neutral
Verfügbarkeit	neutral	neutral
Nachverfolgbarkeit	schlecht	schlecht
Performance	schlecht	schlecht
Sicherheit	schlecht	neutral

Tabelle 3.1: Abschneiden von SOA bei gegebenen Forderungen

Kunden abhängt kommt das Qualitätsattribut Adaptierbarkeit hinzu. Für die Entwickler spielt auch die Testbarkeit des Systems eine große Rolle, in der SOA im Vergleich zur Adaptierbarkeit schlecht abschneidet.

Forderungen	SOA	SOA mit Webservices
Adaptierbarkeit	gut	gut
Testbarkeit	schlecht	schlecht

Tabelle 3.2: Additive Forderungen durch Entwickler und Projektumfeld

Die Nachverfolgbarkeit konnte durch Einsatz einer geeigneten Hosting-Umgebung (Windows AppFabric[4]) welche das Überwachen aller Nachrichten auf dem Enterprise-Service-Bus unterstützt verbessert werden. Für die Testbarkeit des Systems wurden einige Herangehensweisen entwickelt, jedoch gestaltet sich ein automatisierter Systemtest immer noch als schwierig. Als eines der Hauptprobleme bleibt die Performance einer SOA mit WebServices. Die performante Verarbeitung großer Datenmengen stellt in manchen Teilen ein noch nicht gelöstes Problem dar.

[4]Windows Server AppFabric ist eine Sammlung integrierter Technologien, die das Erstellen, Skalieren und Verwalten von Webanwendungen und zusammengesetzten Anwendungen für die Ausführung in IIS vereinfachen.

3.3 Systemarchitektur

3.3.1 Überblick

Die Architektur der CLM-Plattform lässt sich in die bereits in Kapitel 1 vorgestellten Komponenten gliedern. Dienste sind an einen Enterprise Service Bus angeschlossen und können von Nutzern über diesen erreicht werden. Der ESB baut auf SOAP über https um den in den Anforderungen erhobenen Sicherheitskriterien zu genügen. Ein abgekoppelter Teil des ESB benutzt das OPC-Unified Architecture Protokoll um über Dienste mit einem OPC-UA Server zu kommunizieren, welcher die unterschiedlichsten Kontrolleinheiten abstrahiert. Die an den ESB angeschlossenen Dienste lassen sich in die in Kapitel 1.3.2 „Klassifizierung von Diensten“ vorgestellten Service-Typen gliedern: Basis-Daten, Composed und Prozess-Services.

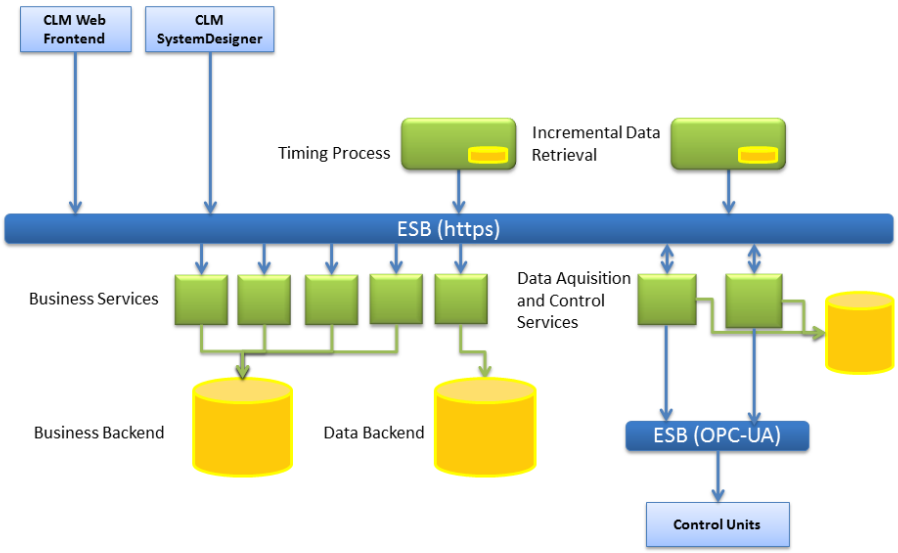

Abbildung 3.2: SOA Architektur der CLM-Plattform

Die CLM-Plattform besitzt drei Backends, auf die Dienste zugreifen. Das Business-Backend hält die Stammdaten der Plattform. Hierbei handelt es sich um Kunden und Anlagendaten, sowie Informationen über den Aufbau einer Anlage, den angeschlossenen Kontrolleinheiten und den im System verfügbaren Datenquellen. Das Data Backend ist ein hochperformanter Datenspeicher, in welchem die ständig aufgezeichneten Messdaten der überwachten Anlagen persistiert werden. Das dritte Backend kann es in mehreren Instanzen geben. Es speichert protokollspezifische Informationen über den Zustand einer Kontrolleinheit, die es ermöglichen Anfragen aus dem System wieder an die richtige physikalische Kontrolleinheit zu binden.

Um reale Beispiele zu den in Kapitel 1.3.2 definierten Service-Typen zu geben werden hier kurz einige Dienste der CLM-Plattform vorgestellt.

3.3.2 Basis-Daten Services

Die Basis-Daten-Services stellen die Grundfunktionalität der CLM-Plattform dar. Sie bieten standard CRUD - Funktionalität (Create, Update, Delete) auf Businessobjekte, aber auch spezifische Schnittstellen für häufig gebrauche Operationen. Beispiele für Basis-Daten-Services der CLM-Plattform sind:

CRUD Services
Stellen die Grundoperationen Create, Update und Delete auf Businessobjekte zur Verfügung. Bei Businessobjekten handelt es sich um domänenspezifische Entitäten wie „Kunde“, „Benutzer“, „Anlage“, „Steuereinheit“ oder „Datenquelle“. Wie auch bei anderen Diensten aber hier im Besondernen ist auf die Sicherheit zu achten. Dies betrifft die Zugriffssicherheit sowie die Transaktionssicherheit dieser Operationen.

Historical Data Series Service
Der Historical-Data-Series Service ist ein typisches Beispiel eines Basis-Daten-Services. Er nimmt die Messdaten aus dem Backend und bereitet sie im zeitlichen Kontext auf. Stellt Methoden zur Verfügung mit welchen eruiert werden kann, für welche Datenquellen und welche Zeiträume Messdaten vorhanden sind.

Data Source Service
Dieser Dienst gibt Zugriff auf alle Datenquellen, die im System einem bestimmten Businessobjekt zugeordnet sind. Dabei muss es sich nicht zwingend um Messdaten handeln, Datenquellen können z.B. auch Zugriff auf Berechnungsergebnisse geben. Im Service-Contract findet man hier typische Get- und Find-Funktionalität.

Persistance Service
Der Persistance Service nimmt Messdaten zur Persistierung im Daten-Backend entgegen und synchronisiert die real auf einer Kontrolleinheit existierenden Messgrößen mit im System zur Verfügung stehenden Datenquellen. Bei diesem Daten-Service ist die Performance besonders wichtig, da, sofern eine Kontrolleinheit zwischenzeitlich vom System getrennt war, eine große Datenmenge auf einmal eintreffen kann.

Live Data Service
Hierbei handelt es sich um einen sehr schlanken Service, dessen einzige Aufgabe es ist für jede Datenquelle den aktuellsten Wert in einem Cache zu halten, damit Anfragen so schnell wie möglich bearbeitet werden können.

3.3.3 Composed Services

Composed Services setzen ihre Funktionalität aus Basis-Daten-Services zusammen. Im System gibt es verhältnismäßig wenige Composed Services, da diese Indirektion, also die SOAP-Serialisierung und Deserialisierung, die Performanz stark beeinträchtigt.

Harvester Control Service
Er dient zum Starten neuer Aufzeichnungsinstanzen für Messdaten und bedient sich dabei der Funktionalität der Basis-Daten-Services des Business-Backends sowie Services der Kontrolleinheit um sein eigenes Backend mit Informationen zu versorgen. Da dieser Dienst ein eigenes Backend besitzt, dessen Informationen aber hauptsächlich nur für den Dienst selbst relevant sind, handelt es sich nicht um einen reinen Composed-Service. Sieht man dieses Backend als Zustand könnte man diesen Service als Prozess-Service oder sofern man der Interaktion mit Basis-Services weniger Bedeutung zukommen lässt, als Basis-Service sehen.

Command Router Service
Der Command Router Service nimmt Anfragen zum Manipulieren einer Anlage entgegen, benutzt die Basis-Daten-Services um Sicherheitsüberprüfungen vorzunehmen und die Anfrage aufzuzeichnen. Danach leitet er die Anfrage an die Zielkontrolleinheit weiter. Damit ist der Command Router einer der wenigen echten Composed-Services der CLM-Plattform. Die Überlegungen an die Performanz sind hier nicht ausschlaggebend da die Sicherheit und nicht die Geschwindigkeit bei der Manipulation einer Anlage ist.

3.3.4 Prozess Services

Prozess-Dienste gelten als länger laufende Dienste, die einen Zustand über mehrere Service-Calls erhalten. Ein oft erwähntes Beispiel ist das des Warenkorbs, in welchen ein Benutzer über Service-Aufrufe Artikel hinzufügt und die er, auch nach Tagen, immer noch im Warenkorb findet, obwohl er den Vorgang zuvor unterbrochen hat. Die Arbeit an der CLM-Plattform hat aber gezeigt, dass es auch für weitaus kürzere Zeitspannen (kleiner als eine Minute) oft sinnvoll ist in Prozess-Diensten zu denken. Die unten angeführten Dienste sind Beispiele für solche Prozess-Dienste, die in der Praxis nie länger als eine Minute laufen.

Incremental Data Retrival Service
Dieser Dienst entsteht aus der Notwendigkeit den sonst zustandslosen Basis-Daten-Services einen solchen hinzuzufügen. Will man für eine Datenverarbeitungsaufgabe eine große Menge an Daten anfordern, ist ein einzelner zustandsloser Service-Call nicht geeignet, da eine einzelne SOAP Nachricht nicht beliebig groß werden kann und Dienste bestimmte Timeouts erfüllen müssen. Dieser Dienst erlaubt das inkrementelle Abrufen von Daten, wobei

der Zustand, welche Daten bereits abgerufen worden sind, über Serviceaufrufe für einen bestimmten Nutzer erhalten bleibt.

Scheduling Service
Der Scheduling Service ist ein länger laufender Prozess-Dienst, welcher über Basis-Daten-Services die von ihm durchzuführenden Jobs erhält, ihren zeitlichen Kontext prüft und gegebenenfalls ausführt. Dabei kann es sich um Regelauswertungen handeln wie z.B.

- Wenn Temperatur des Vorlaufs größer als die des Rücklaufs über mindestens 5 Minuten benachrichtige Benutzer A per SMS
- Länger laufende Batch-Jobs die auf Daten des Daten-Backends operieren.

3.3.5 Service Nutzer

Wie bereits oben gezeigt nutzen sich die Dienste der CLM-Plattform untereinander, die wahren Endnutzer stellen aber zwei Client-Systeme dar, die Zugriff auf die Plattform ermöglichen.

CLM WebControl Das CLM WebControl ist ein Silverlight-Client, welcher es Benutzern ermöglicht über das Internet ihre Anlangen und deren Daten einzusehen. Dieses Frontend präsentiert das im System Designer erzeugte Anlangenschema, welches um Live-Daten angereichert wird und ermöglicht den Status der Anlage einzusehen und direkt Einfluss darauf zu nehmen. Es befasst sich außerdem mit der visuellen Repräsentation von historischen Anlagedaten.

Ein Eindruck von den Frontends, die hier als Servicenutzer auftreten, findet sich in Form von Screenshots im Anhang A.3.

CLM System Designer Der System Designer bildet die Steuerzentrale, über welche Anlagen erzeugt und beschrieben werden können. Er umfasst eine Menge an Verwaltungstätigkeiten wie das Anlegen von Systemregeln, Verlaufsdiagrammen sowie Benutzergruppen und Zugriffsrechten auf Daten. Für diese Aufgaben nutzt der System Designer einen Großteil der von der CLM-Plattform zur Verfügung gestellten Dienste.

3.3.6 Service Host

Die Funktionalität eines Applikationsservers der CLM-Plattform wird durch Services bereitgestellt. Services sind Komponenten, die ihrerseits aus einer Menge von Service Funktionen bestehen, welche einer Client-Anwendung über ein Netzwerk zur Verfügung stehen. Eine Service Komponente ist von sich aus noch nicht funktionsfähig, sondern

wird erst durch Einbettung in eine Laufzeitumgebung, dem Service Host, aktiviert. Dieser Service Host stellt die gesamte notwendige Infrastruktur für Services zur Verfügung, sodass ein Service selbst als rein funktionale Komponenten betrachtet werden kann. Der Service Host schließlich ist eine Anwendung die direkt auf dem Betriebssystem des Applikationsservers aufsetzt.

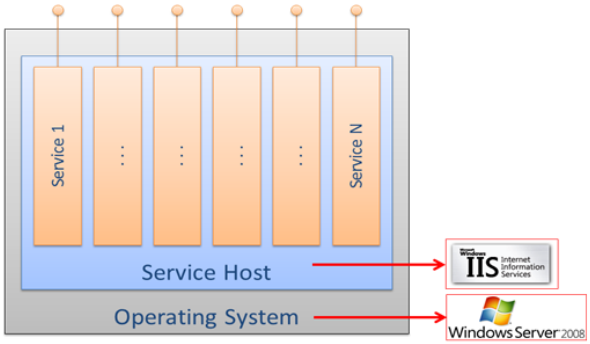

Abbildung 3.3: Hosting-Ungebung mit Windows AppFabric

Es ist nicht notwendig, dass alle Dienste im selben Service Host laufen. Mehrere Service Hosts können über Maschinen verteilt sein. Die macht verteilte Hardwarearchitekturen möglich und lässt einfach Last auf mehrere Maschinen verteilen.

3.4 Softwarearchitektur

Kapitel 2 „SOA und Web Services“ stellt die grundlegenden Standards, SOAP, WSDL, UDDI und die sogenannten WS-* Standards, vor. Für diese gibt es natürlich mehrere Implementierungen wie Java Web Service, Python Web Services und die Windows Communication Foundation. Aufgrund der vielversprechenden Reviews und der .NET-Erfahrung des Entwicklerteams fiel die Entscheidung auf Letzteres. Einen Vergleich zwischen WCF und JAX-WS, dem Gegenüber aus der Java-Welt, finden Sie in Kapitel A.1.

Obwohl theoretisch möglich, ist eine eigenständige Umsetzung der Standards nie in Betracht gezogen worden. Die WS-* Standards sind sehr umfangreich und eine eigene Implementierung würde keine Vorteile bieten, geschweige sie wäre im zeitlichen und finanziellen Rahmen umsetzbar. Auch um die Interoperabilität wirklich gewährleisten zu können ist von einer eigenen Umsetzung oder auch nur einer Anpassung schwer abzuraten.

3.4.1 Windows Communication Foundation

WCF ist ein Software Development Kit für das Entwickeln und Deployment von Webservices unter Windows. Es stellt eine Laufzeitumgebung zur Verfügung die es erlaubt CLR (Comman Language Runtime) Typen als Service bereit zu stellen und andere Dienste als CLR-Typen zu nutzen. Die Webserviceimplementierung von Microsoft hält sich dabei an die Industriestandards und garantiert die Interoperabilität von Diensten. Das erste Release von WCF wurde mit .NET 3.0 ausgeliefert und hat sich kontinuierlich bis zur aktuellen .Net-Version (4.0) weiterentwickelt. Die Kernelemente zur Umsetzung von Webservices mit WCF stellen die Definition von Service-Contracts, Bindings und Endpoints dar. Diese Definitionen übersetzen sich nach außen hin in das den Service beschreibende WSDL-Dokument, welches bereits in Kapitel 2.2 „WSDL - Web Services Description Language“ diskutiert wurde, sowie die Konfiguration des Service-Hosts und bildet die Grundlage der Serviceimplementierung.

Contracts

In WCF stellen alle Dienste sogenannte Contracts zur Verfügung. Dabei handelt es sich um eine plattformunabhängige Beschreibung des Dienstes (WSDL). WCF kennt dabei vier Arten von Contracts, die sich auch im WSDL-Beispiel aus Kapitel 2.2 wiederfinden lassen.

Service-Contract
Beschreibt, welche Operationen der Dienst anbietet. Dabei handelt es sich um eine Interface, welches mit dem Service-Contract-Attribut versehen wird und dient gleichzeitig als Dienstbeschreibung wie auch als Interface für seine Implementierung.

Data-Contract
Ein Data-Contract ist das WCF-Gegenstück zur WSDL-Typendefinition in XSD. Er beschreibt, welche Typen ein Dienst erwartet und zurückliefert. WCF stellt Contracts für alle .NET Basistypen zur Verfügung, für eigene komplexe Typen (Datenklassen) müssen eigene Data-Contracts erstellt werden. Dies ist einfach möglich, indem man die Datenklassen mit dem Data-Contract-Attribut und die benötigten Member mit dem Data-Member Attribut markiert.

Fault-Contract
Dieser Vertrag definiert Fehler, die ein Dienst auslösen kann und wie diese zum Servicenutzer weitergeleitet werden sollen. Im Wesentlichen handelt es sich um das Webservicegegenstück zu einer Exception.

Message-Contract
Message-Contracts erlauben es direkt mit Nachrichten zu interagieren, werden aber nur beötigt um Interoperabilitätsprobleme zu lösen indem direkt in eine SOAP-Nachricht eingegriffen wird. Das ist ein sehr unüblicher Fall und wurde für die Entwicklung der CLM-Plattform auch nicht eingesetzt.

Bindings

´ Bei der Interaktion mit Diensten gibt es verschiedene Aspekte zu berücksichtigen. Etwa, ob die Kommunikation synchron oder asynchron erfolgen soll, ob die Nachrichten sofort zugestellt oder in eine Warteschlange aufgenommen werden sollen. Des Weiteren, welches Transportprotokoll, wie z.B. HTTP/HTTPS, TCP, IPC oder MSMQ (Microsoft Message Queue) verwendet wird. Bei der Wahl eines Bindings werden auch die Sicherheitseigenschaften eines Dienstes festgelegt, etwa ob Message- und/oder Transportlayer-Security auf die Nachricht angewendet werden soll oder die Art der Autorisierung und Authentifizierung. Speziell die WS-Bindings bieten eine Fülle an Optionen, die in dem in Kapitel 2.4 „WS-* Spezifikationen“ vorgestellten WS-* Standards festgelegt sind. Um diese Menge an möglichen Permutationen an Kommunikationsaspekten überschaubar zu halten definiert WCF ein Set von unterschiedlichen Bindings, die auf bestimmte Szenarien zugeschnitten sind. Es besteht aber die Möglichkeit jedes beliebige Binding selbst zu definieren.

Jedem Dienst muss ein Binding zugeordnet werden. Es spezifiziert für den Dienst also das Transportprotokoll, Message Encoding, Communication Pattern, Reliability, Security, Transaction propagation und Interoperabilität. Zu den gebräuchlichsten Bindings zählen: Webservice-Binding (WS-Binding) Dieses Binding nutzt HTTP bzw. HTTPS als Transportprotokoll und unterstützt die in den WS-* Standarts festgelegten Features.

Basic Binding
Das Basic Binding ist dafür gedacht WCF-Services als legacy ASMX

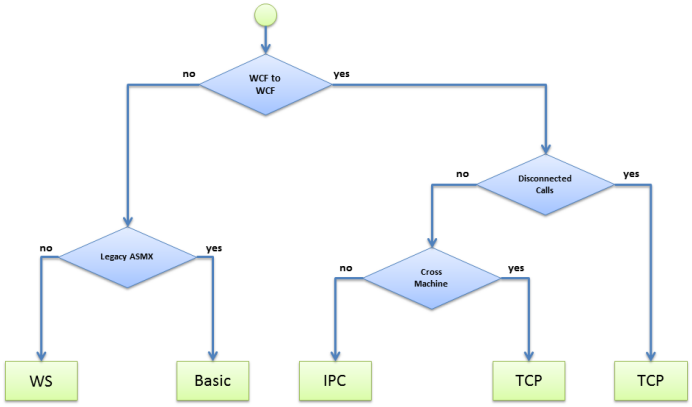

Abbildung 3.4: Entscheidungen zur Wahl des richtigen Bindings

Webservices (ASP.NET Webservices aus dem .NET 1.0 Framework) anzubieten, sodass alte Clients mit dem Dienst arbeiten können.

TCP Binding
Dieses Binding nutzt TCP als Kommunikationsprotokoll und ist für Intrantet-Szenarien gedacht. Es unterstützt eine Vielzahl von Features wie Zuverlässigkeit, Transaktionen, Sicherheit und ist optimiert für WCF-zu-WCF Aufrufe.

IPC Binding
Das IPC (Inter Process Communication) - Binding nutzt Named Pipes als Transportweg. Es unterstützt dieselben Features wie das TCP-Binding, gilt jedoch als sicherer und performanter, da es keinen Zugriff von außerhalb der Hostmaschine erlaubt.

MSMQ Binding
Das MSMQ -Binding (Microsoft Message Queue) nutzt das gleichnamige Protokoll und unterstützt damit „disconnected calls“, legt also Nachrichten in eine Warteschlange, bis sie dem Empfänger auch wirklich zugestellt werden können.

Endpoints

Jeder Dienst besitzt eine Adresse, unter welcher er verfügbar ist, ein Binding, das festlegt, wie mit ihm kommuniziert wird, und einen Service-Contract, der sein Verhalten beschreibt. Diese drei Komponenten werden oft als das ABC (Adress, Binding, Contract) eines Dienstes bezeichnet und ergeben zusammen einen Endpoint.

Um funktional zu sein muss ein Service mindestens einen, kann aber mehrere Enpoints zur Verfügung stellen. Jeder dieser Enpoints muss eine einzigartige Adresse besitzen kann aber gleiche oder unterschiedliche Contracts und Bindings benutzen. Die Definition eines Enpoints kann über Sourcecode erfolgen, wird aber in der Praxis für

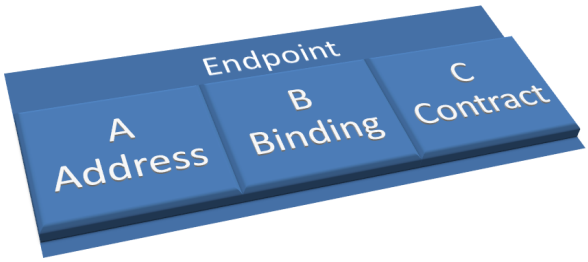

Abbildung 3.5: Die drei Komponenten eines Endpoints

die Hosting-Umgebung konfiguriert. Das heißt, Endpoint und Service-Implementierung sind strikt getrennt.

3.4.2 Implementierung anhand eines Dienstes

Aufgaben des Dienstes

Dieser Abschnitt soll die in Kapitel 3.4.1 vorgestellten Konzepte Service-Contract, Binding und Endpoint anhand eines für die CLM-Plattform entwickelten Dienstes zeigen. Der Dienst, der hier näher beleuchtet wird, hat zwei Hauptaufgaben, die sich mit der Anbindung von logischen Kontrolleinheiten befassen.

Logische Kontrolleinheiten im System sind immer mit der realen Hardware, z.B. einer SPS und dem darauf laufenden OPC-UA Server, verbunden und können alle verfügbaren Mess- , Status- und Kontrollvariablen aus der OPC-Struktur lesen. Aufgabe der logischen Kontrolleinheit ist es, das protokollabhängige Datenmodell, wie das hierarchische Node-Model von OPC, in das abstraktere und protokollunabhängige Modell der CLM-Plattform zu übertragen. Da sich das Datenmodell ändern kann, indem z.B. ein Techniker neue Sensoren anschließt und möglicherweise andere entfernt, wird ist es von Zeit zu Zeit nötig diese Datenmodelle wieder zu synchronisieren. Dies gilt natürlich auch, wenn die Kontrolleinheit das erste Mal an das System angeschlossen wird und ist Aufgabe der Serviceoperation *SynchronizeAllDataSources*. Sind die Datenmodelle von CLM-Plattform und Kontrolleinheit synchron, werden logische Kontrolleinheiten laufend Messdaten an diesen Dienst versenden. Um die Systemlast hier gering zu halten werden nur Messdatenänderungen, die über einem gewissen Hysteresewert liegen, versendet. Logische Kontrolleinheiten benutzen die Serviceoperation *BulkWriteDataValues* um ganze Datenblöcke an das System zu versenden. Der hier beschriebene Dienst ist also die Hauptanlaufstelle für logische Kontrolleinheiten im System. Neben den beschriebenen komplexeren Aufgaben besitzt der Dienst weiters Methoden den Status dieser Kontrolleinheiten an das System zu melden.

Servicedefinition für die Hosting-Umgebung

Einstiegspunkt für die Implementierung des Dienstes ist seine Beschreibung in der SVC-Datei[5], über diese der Dienst für die Hosting-Umgebung beschrieben wird und im Web-Root der Des IIS[6] zu liegen kommt.

Listing 3.1: Konfiguration der SVC-Datei.

```
1 <%
2     @ ServiceHost
3     Language="C#"
4     Debug="false"
5     Factory="Nte.Clm.Server.Hosting.UnityServiceHostFactory"
6     Service="Nte.Clm.Server.Services.Persistance.HarvesterPersitanceService"
7 %>
```

Über die URL dieser Datei kann der Dienst alternativ zur URL des WSDL-Dokuments angesprochen werden. Wie in Abschnitt 3.3.6 „Service Host“ beschrieben handelt es sich bei der Hosting-Umgebung um Windows Server AppFabric, welche eine Erweiterung des Internet Information Servers darstellt und zum Hosten und Überwachen von .NET Diensten entwickelt wurde. Deshalb bezieht sich die Sprachdefinition aus Zeile 3 des Listings 3.1 auch nur auf .NET-Sprachen. Zeile 4 definiert, dass keine Debug-Informationen über den Endpoint des Dienstes nach außen gegeben werden dürfen. Das Aktivieren dieser Funktion ist für die Entwicklung und das Testen von Diensten äußerst wichtig, da sie erlaubt mehr Informationen über Fehler an die Dienstnutzer weiterzugeben, was natürlich im Produktivbetrieb aus Sicherheitsgründen zu vermeiden ist. Zeile 5 definiert, welche Factory von AppFabric zum Instantiieren von Diensten genutzt werden soll. Im Regelfall wird diese Option nicht angegeben und die Standard-Service Host Factory benutzt. Warum es hier nötig war eine eigene Implementierung einzusetzen, wird in Abschnitt 3.4.3 näher erläutert. Die letzte Definition gibt an, welche Serviceimplementierung zur Verfügung steht. Wichtig ist hier zu bemerken, dass hier noch keine Contracts, Bindings oder Endpoints beschrieben wurden, sondern es sich lediglich um Metainformationen für die Hosting-Umgebung handelt.

3.4.3 Definition von Address, Contract und Binding

Endpoint

Um einen Endpoint definieren zu können muss eine Adresse, ein Contract und ein Binding einer Serviceimplementierung zugeordnet werden. Listing 3.2 zeigt die Definition eines Endpoints welche in der Web.config der Serviceapplikation[7] zu liegen kommt. In dieser Konfigurationsdatei sind in Bezug auf Webservices nur die Einträge

[5] Die Dateiendung .svc steht für *Service*.

[6] Internet Information Services (IIS) (vormals Internet Information Server) ist eine Diensteplattform der Firma Microsoft für PCs und Server. Über sie können Dokumente und Dateien im Netzwerk zugänglich gemacht werden. Als Kommunikationsprotokolle kommen hierbei zum Einsatz: HTTP, HTTPS, FTP, SMTP, POP3, WebDAV und andere [Wik10b].

[7] Im IIS und AppFabric werden Webseiten oder eine Sammlung von Diensten zu einer Applikation zusammengefasst

unter dem Abschnitt *system.serviceModel* von Bedeutung. Dieser gliedert sich in die Unterpunkte Services, Bindings und Behaviours (Behaviors sind Komponenten, die im WCF-Kommunikationsstrom bestimmte „Verhaltensweisen“ des Dienstes implementieren. Dies können einfache Dinge wie Komprimierung des Datenstroms sein, aber auch komplexe Erweiterungen wie z.B. Authentifizierung oder Verschlüsselung.)

Listing 3.2: Konfiguration der SVC-Datei.

```
1  <service name="Nte.Clm.Server.Services.Persistance.HarvesterPersitanceService"
       behaviorConfiguration="CustomAuthentication">
2      <host>
3          <baseAddresses>
4              <add baseAddress="https://ntetfslab01.nte.local:444/SolarCheckerLive/SystemServices/" />
5          </baseAddresses>
6      </host>
7      <endpoint
8          name="secureEndpoint"
9          address="HarvesterPersistance"
10         bindingNamespace="http://ntesystems.at/clm/v2/"
11         binding="customBinding" bindingConfiguration="basicSecureBinding"
12         contract="Nte.Clm.Shared.ServiceContracts.Persistance.IHarvesterPersistanceService">
13     </endpoint>
14     <endpoint
15       address="mex"
16       binding="mexHttpsBinding"
17       contract="IMetadataExchange" />
18     </service>
```

Listing 3.2 zeigt die Definition der zwei Endpoints des HarvesterPersistanceServices. Bei ersterem handelt es sich um den eigentlichen Serviceendpoint, der zweite ist der sogenannte Metadata Exchange Endpoint der die Servicebeschreibung anhand des WSDL-Dokuments zugänglich macht. In geschlossenen Systemen ist es wünschenswert diese Informationen nicht öffentlich weiterzugeben, in diesem Fall müssen Dienstnutzer auf anderem Wege zur Serviceinformation, wie z.B. über eine Shared-Library, kommen.

Der hier definierte Service-Endpoint wird durch einen eindeutigen Namen identifiziert. Die definierte Adresse wird der für dem Dienst zugewiesenen Basisadresse angefügt und ergibt zusammen somit eine eindeutige Adresse für den Endpoint auch wenn es mehrere Endpoints, z.B. einen für Kommunikation über TCP und einen für HTTP, gibt.

Als Binding-Typ wird hier Custom-Binding gewählt dessen konkrete Ausprägung unter dem Namen „basicSecureBinding“ definiert ist und in Abschnitt 3.4.3 genauer diskutiert wird.

Der Eintrag „Contract“ verweist auf das Interface welches den Service-Contract definiert und wird in Abschnitt 3.4.3 genauer vorgestellt.

Binding

Wie bereits in Abschnitt 3.4.1 diskutiert können Szenarien auftreten in denen die Standard-Bindings nicht ausreichen. Da ein Frontend der CLM-Plattform auf Silverlight[8] basiert und Silverlight-Applikationen nicht mit WS-Bindings umgehen können,

[8]Microsoft Silverlight ist eine Erweiterung für Webbrowser, die die Ausführung von Rich Internet Applications ermöglicht. Eine Rich Internet Application ermöglicht dem Besucher einer Website z. B. Drag and Drop, 3D-Effekte, Animationen und Unterstützung diverser Videoformate und anderer Medien. [Wik10c]

das basicHttp-Binding aber nicht die benötigten Sicherheitsfeatures unterstützt, musste ein eigenes Binding definiert werden.

Listing 3.3: Bindingkonfiguration

```
<customBinding>
    <binding name="basicSecureBinding"
        openTimeout="01:00:00"
        closeTimeout="01:00:00"
        receiveTimeout="01:00:00"
        sendTimeout="01:00:00">

        <security authenticationMode="UserNameOverTransport">
            <localClientSettings maxClockSkew="23:00:00" />
       <localServiceSettings maxClockSkew="23:00:00" />
       <secureConversationBootstrap/>
        </security>

        <textMessageEncoding messageVersion="Soap11">
            <readerQuotas maxDepth="1000"
                maxArrayLength="104857600"
                maxBytesPerRead="10485760"
                maxStringContentLength="104857600" />
        </textMessageEncoding>

        <httpsTransport maxReceivedMessageSize="32000000" />
    </binding>
</customBinding>
```

Listing 3.3 zeigt die Definition des Binding. Wichtig sind hier die Punkte *Security* und *TextMessageEncoding* die Sicherheitseinstellungen für das Binding festlegen deren Implementierungen in WCF den in Kapitel 2.4 vorgestellten WS-* Spezifikationen entsprechen.

Behaviour

Behaviours stellen in WCF die Möglichkeit dar Diensten spezielles Verhalten hinzuzufügen. Da dies in einer eigenen Definition passiert kann diese für mehrere Dienste einfach wiederverwendet werden.

Listing 3.4: Behavior Definition

```
<behavior name="CustomAuthentication">
  <serviceMetadata httpGetEnabled="False" httpsGetEnabled="True"/>
  <serviceDebug includeExceptionDetailInFaults="False"/>
  <serviceCredentials>
     <!-- Custom Validation Mode -->
     <userNameAuthentication
      userNamePasswordValidationMode="Custom"
      customUserNamePasswordValidatorType="Nte.Clm.Server.Modules.Security.UserAuthentication,
                                           Nte.Clm.Server.Modules.Security"/>
     <!-- Certificate for AppServer-->
     <serviceCertificate
      findValue="NBNTE04"
      storeLocation="LocalMachine"
      storeName="My"
      x509FindType="FindBySubjectName"/>
     </serviceCredentials>
</behavior>
```

Der WS-Security Standard legt fest wie Benutzer-Credentials in SOAP-Nachrichten codiert aber natürlich nicht wie diese validiert werden. Der Abschnitt *userNameAuthentication* legt fest welches Modul dafür zuständig ist und in welcher .Net-Assembly es sich befindet. Diese Klasse muss von *UsernamePasswordValidator* erben und überprüft die Benutzer-Credentials mit den Informationen aus der Datenbank der CLM-Plattform. Der Absatz *serverCertificate* gibt den Ort an, wo das Zertifikat zu finden ist, das die Schlüssel enthält, die zur im Binding definierten Nachrichtenverschlüsselung eingesetzt werden sollen.

Contract

Wie bereits in Kapitel 3.4.1 erwähnt handelt es sich bei einem Dienstvertag um ein Interface, welches um die Attribunte *ServiceContract* und *OperationContract* angereichert wird. In der Praxis sind jedoch noch weitere Attribute zu beachten. Die Angabe eines Namens für eine Operation hält den Contract stabil auch wenn sich der Name der implementierten Methode intern ändert und löst das Problem, dass Operator-Overloading (Hat die Serviceimplementierung etwa zwei Methoden für Create, eine für Objekt A und eine für B müssen diese in z.B. CreateA und CreateB aufgelöst werden) für Servicecontracts nicht eingesetzt werden kann.

Listing 3.5: Behavior Definition

```
1  namespace Nte.Clm.Shared.ServiceContracts.Persistance
2  {
3    [ServiceContract(Namespace="http://ntesystems.at/clm/v2/", Name = "IHarvesterPersistanceService")]
4      public interface IHarvesterPersistanceService
5      {
6          [OperationContract(Name = "SynchroizeAllDataSources")]
7          [FaultContract(typeof(ScFault))]
8          [FaultContract(typeof(ScDataAccessFault))]
9          [FaultContract(typeof(ScDataManipulationFault))]
10         [FaultContract(typeof(ScOperationalFault))]
11         List<Dto.ControlUnitDataSource> SynchronizeAllDataSources(List<Dto.ControlUnitDataSource>
               listOfDataSources, Dto.ControlUnit controlUnit);
12
13         [OperationContract(Name = "BulkWriteDataValues")]
14         [FaultContract(typeof(ScFault))]
15         [FaultContract(typeof(ScDataManipulationFault))]
16         [FaultContract(typeof(ScOperationalFault))]
17         void BulkWriteDataValues(List<Tuple<Dto.ControlUnitDataSource, List<Dto.UntypedData>>>
               dataBulk, Dto.ControlUnit controlUnit);
18
19         [OperationContract(Name = "ReportControlUnitStatus")]
20         [FaultContract(typeof(ScFault))]
21         [FaultContract(typeof(ScDataManipulationFault))]
22         Dto.ControlUnit ReportControlUnitStatus(Dto.ControlUnit controlUnit, Dto.ControlUnitStatus
               status);
23
24         [OperationContract(Name = "SetControlUnitSecurityMode")]
25         [FaultContract(typeof(ScFault))]
26         [FaultContract(typeof(ScDataManipulationFault))]
27         Dto.ControlUnit SetControlUnitSecurityMode(Dto.ControlUnit controlUnit,
               Dto.ControlUnitSecurityMode mode);
28     }
29 }
```

Fällt eine Exception in der Serviceimplementierung, wird diese zum Client hin über SOAP als sogenannter Fault übertragen. Diese Faults beinhalten aber kaum Information über den aufgetretenen Fehler. Darum ist es ratsam eigene Faults zu definieren. Welche Faults eine Operation werfen kann, muss im Servicecontract beschreiben werden.

Data-Contract

Wie bereits in Listing 3.6 gesehen verwenden die Operationen des Dienstes unter anderem die Datenklasse Dto.ControlUnitDataSource. Die Abkürzung Dto steht für Data-Transfer-Object[9]. Wie man an diesem Datacontract sehen kann, sind Vererbun-

[9]Das Transferobjekt oder Datentransferobjekt (Abk.: DTO) ist ein Entwurfsmuster aus dem Bereich der Softwareentwicklung. Es bündelt mehrere Daten in einem Objekt, sodass sie durch einen einzigen Programmaufruf übertragen werden können. Transferobjekte werden in verteilten Systemen eingesetzt, um mehrere zeitintensive Fernzugriffe durch einen einzigen zu ersetzen. Ursprünglich aus [Fow03]

gen möglich und durchaus gebräuchlich. Der Datacontract ControlUnitDataSource erweitert den DataSource-Contract lediglich um zwei Member, die serialisierten Persistierungsregeln die von der logischen Kontrolleinheit ausgeführt werden, und ein Feld, das angibt ob die Datenquelle auf der Kontrolleinheit aktiv ist.

Listing 3.6: Behavior Definition

```
namespace Nte.Clm.Shared.Dto
{
    [System.Runtime.Serialization.DataContract(
          Namespace="http://ntesystems.at/clm/v2",
          Name="ControlUnitDataSource")]
    public partial class ControlUnitDataSource : DataSource, IEntity
    {
        [System.Runtime.Serialization.DataMember(Order=0, IsRequired=true)]
        public virtual string Rules
        {
            get;
            set;
        }

        [System.Runtime.Serialization.DataMember(Order=1, IsRequired=true)]
        public virtual bool IsActive
        {
            get;
            set;
        }
    }
}
```

Die Attribute Order und IsRequired helfen bei der Versionierung des Datenvertrags. Kommen neue Member hinzu, gibt das Order-Attribut an, in welcher Reihenfolge sie serialisiert werden. Somit können Servicenutzer, sofern die neuen Member nicht als IsRequired markiert wurden, den Service weiter nutzen ohne Kenntnis über den neue Datenvertrag zu haben, denn bei der Deserialisierung ist sichergestellt, dass ankommende Daten die nicht im Vertrag festgehalten sind, ignoriert werden.

Serviceimplementierung

Die Implementierung des Dienstes entspricht nun der Implementierung des Servicecontract-Interfaces. Hier können noch Attribute für das Dienstverhalten wie bereits bei der Definition des Service-Behaviors direkt an die Implementierung angefügt werden.

Listing 3.7: Service Implementierung

```
namespace Nte.Clm.Server.Services.Persistance
{
    [ServiceBehavior(Namespace = "http://ntesystems.at/clm/v2/",
                     InstanceContextMode = InstanceContextMode.PerCall)]
    public class HarvesterPersitanceService : Core.Services.ServiceBase,
         Shared.ServiceContracts.Persistance.IHarvesterPersistanceService
    {
        ...
    }
}
```

Das einzige hier angefügte Attribut ist die Definition des InstanceContextModes, also wann von der Hosting-Umgebung eine neue Instanz des Dienstes erstellt werden soll. Da es sich in diesem Fall um einen typischen zustandslosen Dienst handelt, wurde die Option *PerCall* gewählt. Weitere Möglichkeiten wären *Single*, also dass es für alle Servicenutzer nur eine einzige Serviceinstanz gibt und *PerSession* für eine Instanz für jeden Servicenutzer.

Eingriff in die Service Host Factory

Im Regelfall besitzen Serviceimplementierungen nur einen Defaultkonstruktur. Das ermöglicht, dass die Hosting-Umgebung aufgrund einer eintreffenden Anfrage selbstständig eine neue Serviceinstanz erzeugen kann, denn sie kennt über die SVC-Dateien alle Serviceimplementierungen. Für die CLM-Plattform wurde die Business-Funktionalität in mehrere unabhängige Module gekapselt. Diese Module sind in einem Dependency-Injection-Container[10], dem Unity-Container aus der Microsoft Enterprise Library[11], registriert und sollen in die Serviceinstanzen injiziert werden. Dazu sind aber Parameter im Konstruktor notwendig und die Service-Host-Factory muss Kenntniss über den Container besitzen, aus dem sie die im Konstruktor geforderten Parameter auflösen soll. Dies kann über eine eigene Implementierung der Factory erreicht werden und diese muss, wie in Abschnitt 3.4.2 zu sehen, als die gültige Factory für diesen Dienst registriert werden.

[10]Dependency Injection (DI) ist ein Entwurfsmuster und dient in einem objektorientierten System dazu, die Abhängigkeiten zwischen Komponenten oder Objekten zu minimieren. Dependency Injection ist eine Anwendung des Prinzips der Inversion of Control (IoC), bezieht sich aber nur auf die Erzeugung und Initialisierung von Objekten. Sie kann als Verallgemeinerung der Fabrikmethoden verstanden werden. Die Funktionalität bleibt trotz dieser Kontrollumkehr als Einfügung enthalten. Dadurch ist es einfach möglich, Abhängigkeiten zu erkennen. [Wik10a]

[11]Enterprise Library für Microsoft .NET Framework ist eine Bibliothek mit Anwendungsblöcken. Dies sind modulare Komponenten, die Entwickler bei häufig auftretenden Aufgaben während der Entwicklung unterstützen sollen. Sie enthält ein erweiterbares Framework zur Erstellung robuster, skalierbarer Anwendungen. [pp10]

3.5 Hardwarearchitektur

Im Folgenden werden verschiedene Varianten von Hardwareachitekturen aufgelistet. Alle davon sind mit der Architektur der CLM-Plattform umsetzbar oder bereits umgesetzt, wobei im Einzelfall eine Kosten- und Aufwandsdefinition im Kontext der finanziellen Möglichkeiten sowie technischen Notwendigkeiten zu treffen ist.

3.5.1 Komponenten


Application-Server

Der Applikationsserver bildet die zentrale Schnittstelle der Plattform. Er ist verantwortlich für die Verwaltung von

- Stammdaten (Kunden, Anlagen, Benutzer...)
- Monitoring Daten (kontinuierliches Sammeln und Überwachen der SPS Daten)
- Prozessen (Scheduling, Berechnungen, Datenaufbereitung)
- Sicherheit (Authentifizierung und Autorisierung)


Client

Der Zugriff auf den Applikationsserver erfolgt über Client Anwendungen, über die der Benutzer beispielsweise neue Anlagen erstellen und modellieren kann, diese dann überwachen und auf sie fernwirken.

- CLM.WebClient
- CLM.SystemDesigner


Control Unit

Die Steuereinheit wird repräsentiert durch eine SPS (Speicher Programmierbare Steuerung) oder alternativ durch eine ECU (Electronic Control Unit). Sie kann die Anlage eigenständig auf Basis einer Konfiguration (Programmierung) steuern, bietet jedoch nur eingeschränkte lokale Möglichkeiten der Bedienung.


Database

Der Applikationsserver bedient sich einer SQL Datenbank als Datenspeicher. Hier werden sämtliche Bestandsdaten (Kunden-, Anlagen-, Benutzerdaten) abgelegt, ebenso wie die gesammelten Daten der verwalteten Anlagen, bzw. deren Steuereinheiten.

3.5.2 Varianten

Single Server

Diese Variante stellt den einfachsten Aufbau dar. Der Applikationsserver bildet die zentrale Schnittstelle, an die theoretisch beliebig viele Clients und Steuereinheiten angeschlossen werden können.

Probleme

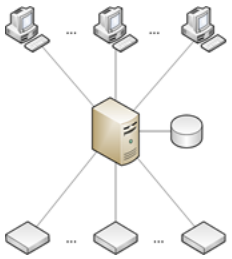

Abbildung 3.6: Hardwarevariante - Single Server

- Die Kommunikationsinfrastruktur (Service Bus) selbst kann insgesamt nur eine begrenzte Anzahl an Verbindungen verwalten, alle darüber hinausgehenden Anfragen werden abgelehnt.
- Die Client Anfragen und die laufenden Prozesse erzeugen am Server eine CPU Auslastung; wenn diese an ihre Grenzen stößt, dann können ebenfalls keine weiteren Anfragen mehr bearbeitet werden.
- Die Client Anfragen und laufenden Prozesse bedingen Zugriffe auf die Datenbank. Der Datenbank Server kann aber nur eine begrenzte Zahl von Anfragen gleichzeitig bearbeiten.
- Wenn der Server ausfällt, dann kommt das einem Totalausfall des Systems gleich (single point of failure) Es kann zu Engpässen in der Kommunikation mit Steuereinheiten kommen, da alle gleichzeitig verwaltet und online gehalten werden müssen.

Multiple Servers

Die nächste logische Erweiterung stellt die Aufteilung der Anlagen innerhalb der CLM-Plattform auf mehrere Serverrechner dar.

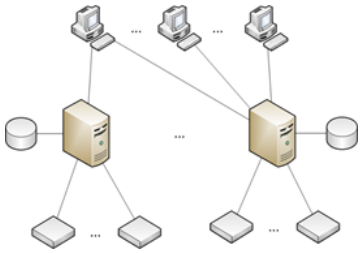

Abbildung 3.7: Hardwarevariante - Multiple Servers

Probleme

- Wenn ein Client Zugang zu allen Anlagen haben möchte, muss er sich zu mehreren Applikationsservern verbinden.
- Wenn ein Applikationsserver ausfällt, dann entspricht das nicht mehr einem Totalausfall, bedeutet aber immer noch, dass eine bestimmte Anzahl von Anlagen nicht mehr verwaltet werden können.

Server Cluster

In dieser Variante werden mehrere Applikationsserver zu einem Verbund (Cluster) zusammen geschlossen, wobei dieser Cluster nach außen wie ein einzelner Applikationsserver auftritt. Der Cluster kann je nach Bedarf um weitere Applikationsserver (Knoten) erweitert und so den gesteigerten Anforderungen angepasst werden.

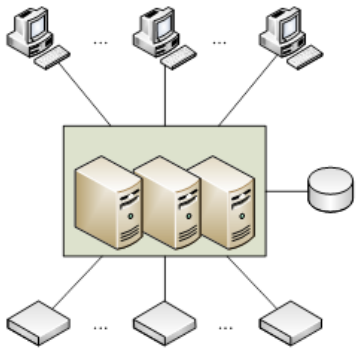

Abbildung 3.8: Hardwarevariante - Server Cluster

Probleme:

- Ein Konzept für einen Applikationsserver Cluster muss ausgearbeitet werden Lastenverteilung notwendig (Software oder Network Load Balancing)
- Eine einzelne Datenbank wird die gesteigerte Last mit der Zeit nicht mehr bewältigen können.

Analog zu einem Applikationsserver Cluster können mehrere Datenbankserver zu einem Datenbank Cluster zusammengefügt werden, der nach außen wiederum wie ein einzelner Datenbankserver auftritt. Die Art der Verwaltung von Datenbank Servern in einem Clusterverbund ist abhängig von der verwendeten Datenbank Plattform (MS SQL, Oracle,)

CLAPP

Ein spezieller Fall tritt dann ein, wenn man die Funktionalität der Plattform für die Überwachung weniger Steuereinheiten oder gar nur für eine einzelne Steuereinheit nutzen möchte. Beispielsweise möchte ein Hausbesitzer aus Datenschutzgründen seine Anlage nicht mit einem Kontrollzentrum verbinden, möchte aber seine Anlage trotzdem überwachen können.

Für diese Situation ist eine kompakte Variante des Applikationsservers vorgesehen, die auf einem durchschnittlich ausgestatteten Haushalts PC lauffähig ist. Dies wird dadurch ermöglicht, dass die Services nicht mehr innerhalb eines IIS untergebracht werden, sondern in einem effizienten und performanten Windows Service. Ein solches Windows Service bietet darüber hinaus noch weitere komfortable Funktionen, wie etwa

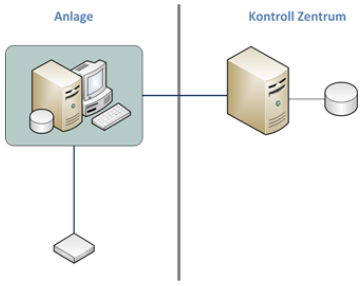

Abbildung 3.9: Hardwarevariante - CLAPP

einen automatischen Neustart und Tracking. Anmerkung: Weil diese Variante die Verschmelzung von Client Anwendung und Applikationsserver vorsieht wird sie als CLAPP bezeichnet. Aber nicht nur aus Datenschutzgründen macht eine solche CLAPP Variante Sinn, auch in einem sogenannten „occasionally connected“ System kann sie von Vorteil sein, also in Situationen, in denen ein permanenter Zugang zum zentralen Applikationsserver nicht möglich ist. Hier ist es sinnvoll, lokal einen Server zu betreiben, der dauernd mit der Steuereinheit verbunden ist, und die Anlage ständig überwachen kann. Von Zeit zu Zeit kann dann dieser Server eine Verbindung zum zentralen Applikationsserver herstellen und seine gesammelten Daten mit diesem synchronisieren. Im Falle eines akuten Problems auf der Anlage kann er sich - falls es die Infrastruktur zulässt - sofort mit dem zentralen Server verbinden und das Problem weiterleiten.

3.5.3 Umsetzungen

Von diesen geplanten Szenarien konnten im Zeitraum dieser Arbeit bereits einige umgesetzt werden. Die Softwareplattform ist grundsätzlich in der Lage die noch fehlenden Szenarien zu bedienen jedoch fehlen für den Betrieb eines Anwendungsclusters noch die Implementierungen von Load Balancing Mechanismen.

Ausbaustufe	Umsetzung	Anmerkung
Single Server	Ja	Voll lauffähig.
Multiple Servers	Ja	Voll lauffähig und zur Zeit die eingesetzte Infrastruktur für die CLM-Plattform.
Server Cluster	Nein	Load-Balancing Mechanismen noch nicht umgesetzt.
CLAPP	Teilweise	Die CLM-Infrastruktur kann gut in einer Offline-Umgebung eingesetzt werden jedoch ist die Re-Syncronisation mit dem Kontrollzentrum momentan nicht mehr im Projektfocus.

Tabelle 3.3: Bereits umgesetzte Hardware-Varianten

3.6 SOA Performance im Monitoring

Im Kapitel 3.2 „Anforderungen“ wurde das Qualitätsmerkmal Performance als Hauptproblem für den Einsatz einer SOA für Monitoring und Steuerung identifiziert. Andere Attribute können leicht durch den Einsatz von Technologien verbessert werden, wie z.B. Nachverfolgbarkeit durch den Einsatz geeigneter Mittelware. Da die schlechte Performance einer SOA aus der Anzahl an Indirektionen und der damit verbundenen Serialisierung und Deserialisierung von Daten in SOAP Nachrichten gründet, ist dem schwerer beizukommen.

In [BM09] wird die Implementierung einer Middelware vorgestellt, die sich die Ähnlichkeit von SOAP Nachrichten zu Nutze macht um den Aufwand des Serialisieren zu verringern und kann damit, je nach Grad der Ähnlichkeit von Nachrichten in einem System, gute Resultate erzielen. In WCF kann die Wahl des richtigen Bindings die Performance stark verbessern, jedoch muss mit diesen optimierten Bindings auf Interoperabilität mit nicht-WCF Diensten verzichtet werden.

Wie in Abschnitt 3.3.3 beschrieben, wurden bei der Implementierung der CLM-Plattform aus diesem Grund für Rohdaten-Dienste, wenn möglich, Indirektionen über Composed-Services vermieden. Da man nicht davon ausgehen kann, dass Firewalls und Router Kommunikation abgesehen von HTTP/HTTPS erlauben, musste auf den Einsatz eines performanteren Bindings verzichtet werden, da Endanwender mit der Konfiguration überfordert wären bzw. Firmenrichtlinen das Freischalten anderer Ports untersagen.

3.6.1 Datenkanäle im System

Abbildung A.4 zeigt die für Messdaten relevanten Datenkanäle. Dieser Aufbau stellt den üblichen Weg dar, wie Daten von einer Steuereinheit (meist SPS) bis zu einem Frontend gelangen und die Zeit für diesen entscheidend für die Aktualität der Daten im Frontend ist.

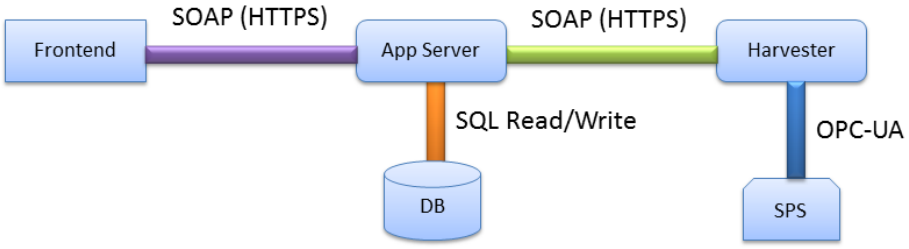

Abbildung 3.10: Kanäle für Messpunktübertragung

Anders als bei vielen SCADA Produkten, die für die Überwachung von Prozessabläufen innerhalb großer Anlagen genutzt werden und sich dadurch innerhalb eines gemeinsa-

men lokalen Netzwerks befinden sind die in Abbildung A.4 gezeigten Komponenten in unterschiedlichen Netzwerken und sind auch geografisch verteilt.

Kanäle

OPC-UA (SPS - Harvester)

Die Kommunikation mit Kontrolleinheiten basiert auf dem OPC-UA Protokoll, der neuesten Weiterentwicklung des OPC Protokolls der OPC Foundation.

> **OPC-UA**
> „The Unified Architecture (UA) is then ext generation OPC standard that provides a cohesive, secure and reliable cross platform framework for access to real time and historical data and events [...] The existing OPC COM based specifications have served the OPC Community well over the past 10 years, but as technology moves on so must our interoperability standards. Here are the factors that influenced the decision to create a new architecture:
>
> - Microsoft has deemphasized COM in favor of cross-platform capable Web Services and SOA (Service Oriented Architecture)
> - OPC Vendors want a single set of services to expose the OPC data models (DA, A+E, HDA ...) OPC Vendors want to implement OPC on non-Microsoft systems, including embedded devices
> - Other collaborating organizations need a reliable, efficient way to move higher level structured data
>
> The Unified Architecture (OPC-UA) is described in a layered set of specifications broken into Parts. It is purposely described in abstract terms and in later parts married to existing technology on which software can be built. [...]“ [Fou10]

Die OPC Foundation stellt jedoch nur den Standard zur Verfügung, leider gibt es noch keine vollständigen Implementierungen dieses Standards. Darum muss, bis voraussichtlich Mitte 2011, auf wichtige Module wie „Historical Data Access“ oder „Alarms and Conditions“ verzichtet werden. Besonders Letzteres ist für die Performance wichtig, da mit Conditions bestimmt werden kann, dass Messwerte nur übertragen werden wenn sich ein Datenwert, unter Berücksichtigung einer angemessenen Hysterese, ändert. Dies bedeutet, dass über diesen Kanal momentan zyklisch alle Datenwerte einer Kontrolleinheit abgefragt werden müssen und erst die Harvester-Komponente nur Datenänderungen weitergeben kann. Dies hat bedeutenden Einfluss auf die Gesamtperformance bei steigender Anzahl von Datenpunkten.

Die Daten für diesen Kanal wurden empirisch gemessen und ergeben ca. 150 Datenpunkte/Sekunde über das öffentliche Netz für eine Kontrolleinheit in Graz (Österreich)

und einem Serverrechner in Deutschland, wobei die Kontrolleinheit über einen UMTS-Datenstick angebunden ist.

HTTPS/SOAP (Harvester - AppServer)

Hierbei handelt es sich um den ersten SOAP Kanal. Da Datenpunkte keine komplexe Struktur aufweisen, werden sie relativ zur Anfrage- und Versendezeit schnell serialisiert. Da es sich bei SOAP jedoch um ein XML basiertes Protokoll handelt, nimmt die Datenmenge schnell zu. Aus ursprünglich 25 Byte Rohdaten für einen Datenpunkt (Wert, Zeitstempel, Typinformation) werden in XML serialisiert ca. 450 Byte Daten.

Datenpunkte	**50**	**100**	**200**	**500**
XML Größe	22 kB	42 kB	88 kB	219 kB
Serialisierung	5 ms	8 ms	13 ms	27 ms
Übertragung Internet	192 ms	343 ms	678 ms	1711 ms
Übertragung LAN	54 ms	79 ms	152 ms	341 ms

Tabelle 3.4: Messwerte für SOAP/HTTPS Kanal

Im Vergleich zur Übertragungszeit dieser Datenmenge fällt die reine Zeit für die Serialisierung sehr gering aus.

Interprocess (AppServer - SQL Datenbank)

Dieser Kommunikationskanal gehört nicht zur SOA-Infrastruktur und wird nur erwähnt, da er eine limitierende Obergrenze für den Datendurchfluss darstellt, sofern die Datenbanken nicht auf mehrere Maschinen verteilt werden (siehe 3.5.2 „Multiple Servers“).

Datenpunkte	**50**	**100**	**200**	**500**
1 Anlage	0,01 sek	0,02 sek	0,03 sek	0,08 sek
5 Anlagen	0,04 sek	0,08 sek	0,17 sek	0,42 sek
10 Anlagen	0,08 sek	0,17 sek	0,33 sek	0,83 sek
50 Anlagen	0,42 sek	0,83 sek	1,67 sek	4,17 sek
100 Anlagen	0,83 sek	1,67 sek	3,33 sek	8,33 sek
200 Anlagen	1,67 sek	3,33 sek	6,67 sek	16,67 sek
500 Anlagen	4,17 sek	8,33 sek	16,67 sek	41,67 sek

Tabelle 3.5: Zeit in Sekunden für die Persistierung von Datenpunkten bei unterschiedlicher Anzahl an Anlagen.

Für einzelne Anlagen ist der OPC-UA und SOAP Kanal zum AppServer parallel, jedoch treffen die Daten mehrerer Anlagen am selben Backend ein. Tabelle 3.5 gibt Aufschluss, wie viele Anlagen und Datenpunkte von einer Maschine bedient werden

können. Obwohl eine Anlage potenziell mehrere tausend Datenpunkte besitzt, sind für die Überwachung und Energiebuchhaltung nur ein Bruchteil relevant, da es sich bei den Übrigen um interne Statusinformation oder Zwischengrößen handelt. In den bereits installierten Anlagen werden immer unter 50 Datenpunkten aufgezeichnet. Größere Anlagenkomplexe können kaskadierte OPC-Server besitzen, welche mehrere SPSen nach außen hin kapseln und so die Menge an Datenpunkten für einen Kanal rasch ansteigt.

HTTPS/SOAP (Frontend - AppServer)

Daten können beim Eintreffen nicht sofort an die Frontends weitergegeben werden, da z.B. Web-Frontends wie Silverlight-Applikationen nicht aktiv beschickt werden können, da sie in ihrer eingeschränkten Laufzeitumgebung natürlich keine Endpoints zur Verfügung stellen können und Dublex-Bindings nicht unterstützt werden. Deswegen müssen Frontend Daten aus dem Backend pollen, was einen Nachrichtenoverhead mit sich bringt. Für Übertragungszeiten und Datenmenge gilt hier dasselbe wie für den ersten SOAP Kanal.

3.6.2 Resultate

Wie aus den vorangegangenen Messwerten leicht ersichtlich ist, trägt der OPC-UA Kanal momentan am meisten zum Zeitverhalten bei (wie aus Abbildung 3.11 ersichtlich im Durchschnitt 61%).

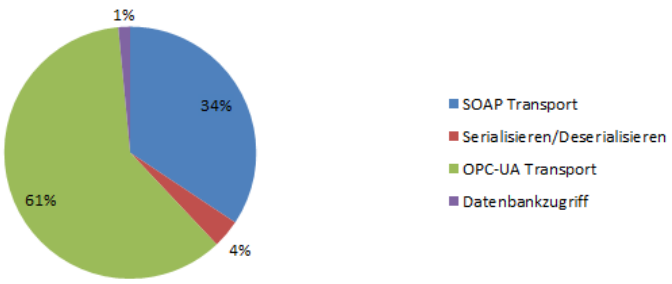

Abbildung 3.11: Anteil am gesamten Zeitverhalten

Sobald es aber möglich wird per OPC-UA nur noch Datenpunktänderungen zu verschicken, reduziert sich die Zeit hier enorm. Dazu muss die Beschaffenheit von Datenpunkten noch einmal genauer beleuchtet werden:

Datenpunkte
Im Bereich des Energiemonitoring handelt es sich bei den meisten relevanten Datenpunkten um Temperaturen, Durchflüsse und Leistungen. Erstere ändern sich nur sehr träge, sodass bei geeigneter Hysterese nur durchschnittlich 1-2 sinnvollen Datenwerten pro Minute zu rechnen ist. Anders verhält es sich mit den Leistungen. Viele Spitzen, z.B. beim Zuschalten eines Geräts,

dauern nur wenige Sekunden an. Diese Spitzen können zwar für eine Energiebuchhaltung vernachlässigt werden, für eine sinnvolle Anlagendiagnose müssen sie aber erfasst werden können.

Erstaunlich ist, dass die Zeit für das Serialisieren der Daten kaum in das Gesamtverhalten mit einfließt und ist wohl darin begründet, dass Listen von Datenpunkten keine komplexen Strukturen sind, so dass erzeugte XML nicht so groß wird.

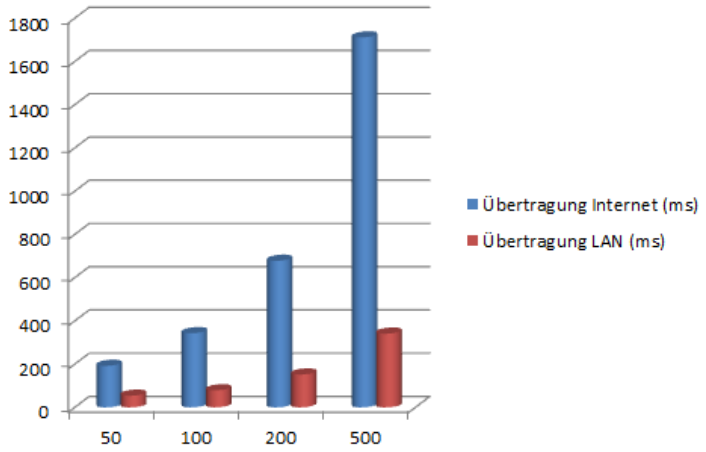

Abbildung 3.12: Gegenüberstellung SOA Indirektion öffentliches und lokales Netz.

Für das Projekt wurde eine Zeitspanne von fünf Sekunden für den Weg bis zum Frontend festgesetzt. Diese Zeit wird im Livebetrieb nie überschritten, aus den Messungen ergibt sich jedoch , dass ca. 500 Datenpunktänderungen/Sekunde und Anlage erreicht werden können, wobei die Gesamtanzahl der aufgezeichneten Datenpunkte aller Anlagen pro Sekunde für eine Servermaschine nicht über 5000 steigen darf. Wegen der Parallelität des eingehenden SOAP-Kanals ist das Zeitverhalten hier optimal, wenn die Anzahl der Datenpunkte gleichmäßig auf alle Anlagen im System verteilt ist.

Kapitel 4

Zusammenfassung

4.1 SOA - Pilotprojekt

Im Rahmen dieser Arbeit wurde versucht ein Verständnis für serviceorientierte Architekturen zu entwickeln und dieses für den Entwurf und Umsetzung der CLM-Plattform einzusetzen. Dieses Projekt kann durchaus als der Pilot aus den vier „P“ von „People, Pilot, Plan, Proceed“ [EP06] gesehen werden. Wie sich jedoch herausstellte, handelte es sich aus mehreren Gründen keineswegs um ein typisches SOA Pilotprojekt:

- **Motivation**
  Die Motivation für den Einsatz von SOA war rein technisch und nicht aus einer Unternehmensphilosophie, also einem „Top-Down“ Ansatz, heraus getrieben. Vielmehr wurde von der technischen Leitung der in [KBS04] beschriebene „Bottom-Up“ Ansatz praktiziert, in dem sich SOA durch erfolgreiche Einzelprojekte im Unternehmen etabliert. Obwohl sich die Entscheidung für SOA im Endeffekt als sehr positiv herausgestellt hat, kann nicht geleugnet werden, dass zu Projektbeginn das in Kapitel 1.9 beschriebene Anti-Pattern „Technology Bandwagon“ aus [JA05] in einer etwas abgewandelten Form gelebt wurde.

- **Team**
  Da das Entwicklerteam zu Projektbeginn keine Erfahrung mit SOA aufweisen konnte, fand erst allmählich eine Bewusstseinsbildung statt. Dies führte anfänglich zum Anti-Pattern „SOA = Webservices“, was bedeutet, dass Webservices wie RPC benutzt werden und Services noch eine falsche Granularität besitzen um von anderen existierenden oder zukünftigen Systemen genutzt werden zu können. Nach [Jos09] kommt dies bei den ersten SOA-Projekten häufig vor, ist aber nicht zwingend als schlecht anzusehen, da es einen notwendigen Schritt in die richtige Richtung darstellt und zu Beginn die Anforderungen für einen vollen SOA Ausbau noch nicht gegeben sind. In [Jos09] wie auch [KBS04] wird beschrieben, dass eine solche Bewusstseinsbildung unumgänglich ist, jedoch weitaus schneller vonstatten geht, wenn das Team über zumindest einen Experten verfügt.

- **IST-Landschaft**
  Der grundlegendste Unterschied bestand in der IST-Landschaft. Typische SOA-Projekte beschäftigen sich mit der Überführung bestehender Systemlandschaften. In einem SOA Pilot sollen eine nicht zu große Menge an Systemen betrachtet und deren Integration, wie in Kapitel 1.8 „Einführen von SOA“ beschrieben, umgesetzt werden. Jedoch darf die Menge an Systemen auch nicht zu klein sein, da sie dann für eine SOA irrelevant wird. Zu Projektbeginn war hier aber noch keines der Systeme vorhanden, sondern nur in einem Projektplan für die nächsten Jahre festgelegt (siehe 3.1.2 Energy Value Systems). Im Projektverlauf sind bis jetzt die geplanten Basis-Systeme entstanden, sodass das Projekt nun eine Größe aufweist die für SOA Relevanz besitzt. Die Entscheidung für ein so untypisches Vorgehen entstand, wie in Abschnitt 3.2.6 „Erwartungen an SOA“ beschrieben, aus der Erwartung heraus, dass eine von Projektbeginn geplante und umgesetzte SOA in Zukunft die Entwicklung von neuen „Energy-Value-Systemen“ und deren Integration beschleunigen soll.

- **Prozess**
  SCRUM als Entwicklungsprozess ist für SOA-Projekte sicher unüblich. Besonders für die Entwicklung der nötigen Infrastruktur und Basis-Services hat sich dieser Prozess aufgrund seiner sehr kurzen Iterationen als ungeeignet erwiesen. SCRUM hat zum Ziel, dass nach jeder dieser Iterationen die entwicklete Software einen produktfähigen Zustand aufweist. Dies ist allgemein für Neuentwicklungen problematisch, kann aber dadurch gelöst werden, dass eine Applikation stetig um kleine Funktionalitäten wächst. Für SOA-Landschaften ist diese Sichtweise zu kurzfristig und resultiert in suboptimalen Lösungen, die immer wieder neu überarbeitet werden müssen, was zu einem Mehraufwand an Entwicklungszeit führt. Steht jedoch die Infrastruktur und Basis-Services, kann SCRUM durchaus gut eingesetzt werden um das System um neue Dienste zu erweitern, da diese gut gliederbare getrennte Entwicklungskomponenten darstellen.

Zusammenfassend hat sich SOA jedoch als ein guter Ansatz erweisen, die Entwicklung von Business-Backend, Frondends und externen „Energy-Value-Systemen“ zu entkoppeln und neue Systeme unkompliziert integrieren zu können. Die Erfahrung des Teams ist steil gestiegen und ermöglicht nun qualitativ hochwertige Lösungen. Wenn hier auch nicht SCRUM in „Reinform“ gelebt werden kann, so wurden jedoch Anpassungen an dem Prozess vorgenommen um eine weitsichtigere SOA-Planung zu ermöglichen.

4.2 Herausforderungen

Das Projekt verlangte ein sehr schnelles Erfassen und Umsetzen von SOA Konzepten. Neben dem konzeptionellen Verständnis für SOA galt es auch sich in die für eine erfolgreiche Umsetzung notwendigen Technologien einzuarbeiten. Dabei handelte es sich nicht nur um Windows Communication Foundation.

Für die Backendentwicklung wurden mehrere komplexe Komponenten benutzt: Entity

Framework 4 für die Datenhaltung im Backend und die Überführung von Daten zu DTOs (siehe Abschnitt 3.4.3 „Data-Contract“), also datenbankunabhängigen Transferobjekten, die in einer SOA genutzt werden können. Dies gestaltete sich dahingehend als schwierig da ORM-Mapper wie EF4 noch immer keine hinreichende N-Tier Fähigkeit besitzen. Ihre Methoden für Change-Tracking gehen von lokalen Applikationen aus und ihre Mechanismen können für verteilte Applikationen nicht funktionieren. Microsoft Enterprise Library für die Validierung von DTO's und das darin inkludierte Prism Framework um Dienste und Dienstlogik über Dependency Injection zu entkoppeln (siehe 3.4.3). Windows AppFabric wurde als Hosting Umgebung eingesetzt (siehe 3.3.6 „Service Host“), deren Konfiguration, sofern sie über „Hello World“ Beispiele hinausgeht und realen Anforderungen genügen muss, keineswegs trivial ist. Als Frontend-Technologie gestaltete sich in Bezug auf SOA besonders Silverlight oft als problematisch, da es aufgrund der für Webapplikation üblichen Restriktion auf Asyncrone Message Exhange Patterns nur ausgewählte Bindings und WS-* Standards unterstützt.

Zusammenfassend haben alle eingesetzten Technologien zur erfolgreichen Umsetzung beigetragen, jedoch ist es zu empfehlen den Einsatz jeder weiteren Technologie genau zu prüfen, da die Komplexität der Gesamtlösung unweigerlich und dauerhaft ansteigt. Ein vorausschauendes Entwickeln ist für SOA Umgebungen natürlich unumgänglich, jedoch sollte das aus Extreme Programming bekannte YAGNI - Prinzip (You Ain't Gonna Need It) [CZ04] im Kopf behalten werden.

4.3 Ausblick

Zum Zeitpunkt des Abschlusses dieser Arbeit werden bereits eine Produktiv- und zwei Testanlagen über die CLM-Plattform überwacht, gesteuert und diagnostiziert. Die Zahl der Produktivanlagen wird in Zukunft stetig ansteigen. Die verteilte Architektur macht es möglich auf mehrere Servermaschinen zu skalieren und hunderte von Anlagen zu bedienen. Des Weiteren wird das Dienstangebot ausgebaut werden um neue „Energy-Value-Systeme“, die sich mit der Thematik Energiebuchhaltung und Diagnose beschäftigen, bedienen zu können. Die größten Herausforderungen für die Zukunft stellen hierbei das Erreichen von hinreichender Performanz für Operationen auf großen Datenmengen dar sowie die Versionierung von Diensten, um gegenüber älteren Dienstnutzern abwärtskompatibel zu bleiben.

Literatur

[Bea07] James Bean. Soa from a-to-z. In *Presented at DAMA International and Wilshire Metadata conference, Boston*, 2007.

[Bea10] James Bean. *SOA and Web Services Interface Design*. The MK/OMG Press, 2010.

[BM09] Parinaz Saadat Behrouz Minaei. Soap serialization performance enhancement, design and implementation of a middleware. *International Journal of Computer Science and Information Security*, 6:6, 2009.

[Cha04] David A. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[Cha09] David Chappell. Introducing windows communication foundation in .net framework 4, 2009.

[Con86] Melvin E. Conway. How do committees invent? In *Datamation*, 1986.

[CZ04] Lowell Lindstrom Carmen Zannier, Hakan Erdogmus, editor. *Extreme Programming and Agile Methods - XP/Agile Universe 2004*. Springer Verlag, 2004.

[Dij72] Edsger W. Dijkstra. Chapter i: Notes on structured programming. *Academic Press Ltd.*, 1:1–82, 1972.

[DJ07] John Evdemon Diane Jordan. Web services business process execution language version 2.0. Technical report, OASIS, 2007.

[DK75] Frank DeRemer and Hans Kron. Programming-in-the large versus programming-in-the-small. In *Proceedings of the international conference on Reliable software*, pages 114–121, New York, NY, USA, 1975. ACM.

[EG94] Ralph Johnson John Vlissides Erich Gamma, Richard Helm. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[EP06] Hugh Taylor Eric Pulier. *Understanding Enterprise SOA*. MANNING, 2006.

[Erl05] Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, 2005.

[Fou10] OPC Foundation. Opc unified architecture. www.opcfoundation.org, 2010.

[Fow03] Martin Fowler. Patterns of enterprise application architecture. Addison-Wesle, 2003.

[Gmb07] InnoQ Deutschland GmbH. Web services standards as of q1 2007. http://www.innoq.com/soa/ws-standards/poster/, 2007.

[Hef07] Randy Heffner. Web services specifications: Core xml specs, 2007.

[JA05] Dr. Mamdouh Ibrahim Jenny Ang, Luba Chebakov. Soa antipatterns - the obstacles to the adoption and successful realization of service-oriented architecture, 2005.

[JB06] Roland Schmelzer Jason Bloomberg. *Service Orient or Be Doomed - How Service Orientation Will Change Your Business*. John Wiley & Sons, 2006.

[Jos09] Nicolai Josuttis. *SOA in der Praxis - System-Design für verteilte Geschäftsprozesse*. dpunkt.verlag, 2009.

[KBS04] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[Ler10] Julia Lerman. *Programming Entity Framework: Building Data Centric Apps with the ADO.NET Entity Framework*. O'Reilly, 2010.

[Löw09] Juval Löwy. *Programming WCF Services*, volume 2. O'Reilly, 2009.

[Mah05] Qusay H. Mahmoud. Service-oriented architecture (soa) and web services: The road to enterprise application integration (eai). Web, 2005.

[Mas10] Dr. Dieter Masak. *Der Architekturreview*. Springer Verlag, 2010.

[Mel10] Ingo Melzer. *Service-orientierte Architekturen mit Web Services*, volume 4. Spektrum Akademischer Verlag, 2010.

[Mic05a] Microsoft. Ubr shutdown faq. http://uddi.microsoft.com/about/FAQshutdown.htm, 2005.

[Mic05b] Sun Microsystems. Javatm api for xml web services annotations. https://jax-ws.dev.java.net/jax-ws-ea3/docs/annotations.html, 2005.

[MLM+06] Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter Brown, and Rebekah Metz. Reference model for service oriented architecture 1.0. Technical report, OASIS, 2006.

[OH10] Andreas Holubek Oliver Heuser. *Java Web Services in der Praxis - Realisierung einer SOA mit WSIT, Metro und Policies*. dpunkt.verlag, 2010.

[Ora10] Oracle. Metro web services overview. http://www.oracle.com/technetwork/java/index-jsp-137004.html, 2010.

[pp10] Microsoft patterns & practices. Microsoft enterprise library. http://msdn.microsoft.com/en-us/library/ff648951.aspx, 2010.

[PRA06] Sam Higgins Paul R. Allen, Paul Allen. *Service orientation: winning strategies and best practices*. Cambridge University Press, 2006.

[Res76] And Human Resources. Nwg/rfc 707 jew 14-jan-76 19:51 34263 ncc 76 a high-level framework for network-based resource sharing the goal, resource sharing 1, 1976.

[RTF02] Richard N. Taylor Roy T. Fielding. Principled design of the modern web architecture. *ACM Trans. Internet Technology*, 2:115–150, 2002.

[W3C02] W3C. Web services description requirements. http://www.w3.org/TR/ws-desc-reqs/, 2002.

[Wik10a] Wikipedia. Dependency injection. http://de.wikipedia.org/wiki/DependencyInjection, 2010.

[Wik10b] Wikipedia. Microsoft internet information services. http://de.wikipedia.org/wiki/Internet

[Wik10c] Wikipedia. Microsoft silverlight. http://de.wikipedia.org/wiki/Microsoft

[Wik10d] Wikipedia. Scrum. http://de.wikipedia.org/wiki/Scrum, 2010.

[Wik10e] Wikipedia. Ws-business process execution language. http://de.wikipedia.org/wiki/WS-BusinessProcessExecutionLanguage, 2010.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Anhang

A.1 Technologievergleich WCF und JAX-WS

Kapitel 2 „SOA und Web Services“ stellt die grundlegenden Standards, SOAP, WSDL, UDDI und die sogenannten WS-* Standards, vor. Diese können natürlich in unterschiedlichen Programmiersprachen umgesetzt werden. Aufgrund des Umfangs und Komplexität ist es aber abzuraten die benötigten Komponenten selbst zu implementieren. Für eine Umsetzung von Webservices wird man immer auf APIs und Libraries der jeweiligen Sprache zurückgreifen. Obwohl es für viele Sprachen Webservice-Implementierungen gibt, sind die meisten nicht besonders ausgereift und für die Implementierung von Enterprisesystemen nicht anzuraten. So kann man beispielsweise einfach über Python einen Webservice ansprechen, jedoch bieten die Bibliotheken und Tools nicht die Infrastruktur für enterprisefähige Servicehosting-Umgebungen und Enterprise Service Bus Lösungen. Weiters wird für effizientes und produktfähiges Entwickeln eine möglichst gute Integration in die jeweilige Entwicklungsumgebung vorausgesetzt. Die derzeit gängigen Sprachen für Enterprisesysteme stellen Microsoft .NET Sprachen wie C# und Java dar. Darum ist es nicht verwunderlich, dass die besten und umfangreichsten Implementierungen für diese Sprachen zu finden sind.

Listing A.1: Beispiel: Einfacher Webserviceaufruf in Python

```
1 from SOAPpy import WSDL
2 server = WSDL.Proxy('/path/to/your/GoogleSearch.wsdl')
3 key = 'YOUR_GOOGLE_API_KEY'
4 results = server.doGoogleSearch(key, 'mark', 0, 10, False, "", False, "", "utf-8", "utf-8")
```

Im Fall von .NET werden Webservices mit der Microsoft Communication Foundation (WCF) realisiert welche in das .NET Framwork integriert ist. WCF ist vollständing in die Microsoft Entwicklungsumgebung Visual Studio integriert[1]. Microsoft bieten mit Windows AppFabric eine Erweiterung für den Internet Information Service (IIS) um ihn um Servicehosting- und Monitoringfunktionalität zu erweitern.

Die Windows Communication Foundation wird von Microsoft wie folgt beschrieben:

[1] ab Visual Studio 2005

> „The move to service-oriented communication has changed software development. Whether done with SOAP or in some other way, applications that interact through services have become the norm. For Windows developers, this change was made possible by Windows Communication Foundation (WCF). First released as part of the .NET Framework 3.0 in 2006, then updated in the .NET Framework 3.5, the most recent version of this technology is included in the .NET Framework 4. For a substantial share of new software built on .NET, WCF is the right foundation.“ [Cha09]

In Java entspricht die Umsetzung von Webservices der Implementierung des Webservice-Stacks Metro. Er umfasst, wie auch WCF, Implementierungen der WS-* Standards, die in Kapitel 2.4 „WS-* Spezifikationen“ beschrieben wurden, weiters Klassen zur Serialisierung und Deserialisierung von Objekten für SOAP-Nachrichten, wobei hier JAX-WS (Java API for XML - Web Services) die API zur Erstellung von Webservices darstellt, JAXB (Java Architecture for XML Binding) zur Serialisierung und Deserialisierung dient und WSIT (Web Services Interoperability Technologies) Implementierungen der WS-* Standards mitbringt. Im Gegensatz zu WCF gibt es in JAVA mehrere Hosting-Lösungen, die für Enterprise-Entwicklungen in Frage kommen, wie z.B. JBoss Enterprise Middleware, GlassFish Application Server oder Apache Tomcat.

Der Metro Webservice-Stack wird von Oracle wie folgt beschrieben:

> „Web services are Web based applications that use open, XML-based standards and transport protocols to exchange data with clients. Web services are developed using Java Technology APIs and tools provided by an integrated Web Services Stack called Metro. The Metro stack consisting of JAX-WS, JAXB, and WSIT, enable you to create and deploy secure, reliable, transactional, interoperable Web services and clients. The Metro stack is part of Project Metro and as part of GlassFish, Java Platform, Enterprise Edition (Java EE), and partially in Java PlatForm, Standard Edition (Java SE).“ [Ora10]

Auf WCF wurde bereits in Kapitel 3.4.1 eingegangen. Nachfolgend wird nun JAX-WS vorgestellt und danach beide in einem Implementierungsbeispiel gegenübergestellt.

A.1.1 JAX-WS

Contract

Als Contract versteht sich allgemein gesehen das WSDL-Dokument. Bei der Entwicklung von Services kann es nun zwei Designrichtungen geben: „Contract First“ oder „Code First“. In Java können beide Varianten angewendet werden, [OH10] merkt aber an, dass es bei „Code First“, wo alle benötigten Artefakte aus der Java-Klasse erzeugt werden, es leicht zu Interoperabilitätsproblemen aufgrund unterschiedlicher Service-Stack Implementierungen kommen kann, da bei der Generierungen Annahmen getrof-

fen werden müssen. Eine weitverbreitete Vorgehensweise, um das WSDL-Dokument nicht selbst erstellen zu müssen, ist mit dem „Contract First“ Ansatz zu beginnen und danach, um den Vertrag stabil zu halten, in den „Contract First“ Ansatz überzugehen.

Wie auch bei WCF arbeitet man in der Java-Welt mit Attributen, hier Annotations, um Klassen um Metainformation anzureichern. Beim „Code-First“ Ansatz werden die Serviceklassen mit der Annotation *@WebService* und Servicemethoden mit *@WebMethod* versehen.[2] Binding und Endpoint werden nicht in einer seperaten Konfiguration definiert, sondern entweder direkt im WSDL-Dokument oder mit Annotationen im Code.

JAX-WS Annotations

Mit Annotations werden Dienstimpkementierungen bzw. deren Interfaces um Metainformation angereichert. Eine vollständige List der Annotaions für JAX-WS findet man unter [Mic05b]. Die wichtigsten werden hier kurz vorgestellt.

- **javax.jws.WebService**
  Der Zweck dieser Annotation ist es eine Webservice-Implementierung oder ein einen Service beschreibendes Interface zu markieren. Alle Serviceimplementierungen müssen die @WebService Annotaion besitzen. Besonders hervorzuheben ist die *endpointInterface* Eigenschaft. Diese zeigt auf ein Interface und sofern gesetzt werden alle weiteren Annotationen für diese Klasse ignoriert und die Annotaions des Interfaces gelten für die Serviceimplementierung.
- **javax.jws.WebMethod**
  Methoden, die mit dieser Annotation markiert sind, werden als Servicemethoden in das WSDL-Dokument übernommen. Es werden Operationsname (dieser kann vom Methodennamen abweichen) und WSDL-Namespace definiert.
- **javax.jws.OneWay**
  Definiert eine Operation als „One-Way“-Operation, d.h., es wird keine Antwortnachricht erwartet.
- **javax.jws.WebParam**
  Diese Annotation erlaubt es das Mapping von Methodenparameter zu Nachrichten-Parameter zu beeinflussen. Damit ist es also leicht möglich einen Parameter im WSDL-Dokument unter einem anderen Namen bekannt zu machen als in der Implementierug.

Insgesamt gibt es ca. 30 Annotationen im Zusammenhang mit Webservices, einige davon werden nur beim „Contract-First“-Ansatz vom Codegenerator eingesetzt und sind nicht für Entwickler gedacht.

[2] @WebService entspricht also ServiceContract und @WebMethod dem OperationContract Attribut.

A.1.2 Vergleich

In Java gibt es für alles eine eigene API. JAX-WS bietet eine sehr gute Umgebung zum Implementieren von SOAP-basierten Webservices und Web Service Clients. JAX-WS unterstützt hier hauptsächlich SOAP/HTTP, also das Standard Webservice-Szenario. Möchte man aber eine optimierte Kommunikation für z.B. Java-zu-Java Anwendungen oder Interprozesskommunikation, benötigt man eine andere API. In der .Net-Welt vereinigt WCF verscheidenste Arten der Kommunikation (Siehe Bindings in Abschnitt 3.4.1) und hat die dort ehemals präsenten APIs wie .NET Remoting und ASP.NET Webservices ersetzt. Implementierung und API unterscheiden sich nicht bei .NET-zu-.NET IPC Kommunikation oder .NET-zu-JAVA HTTP Kommunikation. Dafür bietet WCF unterschiedliche Bindings an, die entkoppelt vom Servicecode, Transportprotokoll, Message Encoding, Communication Pattern, Reliability, Security, Transaction propagation und Interoperabilität für den Dienst festlegen. Diese Möglichkeiten existieren theoretisch auch in JAVA ,erfordern aber weit mehr Arbeit und Kenntnisse.

In Java werden Services als einfache Javaobjekte[3] erzeugt und einem Java Applikationsserver übergeben, welcher diese in Laufzeitkontainer einbettet. Wie bereits zu Beginn von Kapitel A.1 erwähnt gibt es als Applikationsserver hier mehrere Möglichkeiten, die Art des Hostings bleibt aber dieselbe. WCF bietet mehr unterschiedliche Optionen: IIS mit AppFabric Erweiterung kann als Applikationsserver benutzt werden, Services können als Windows-Service laufen oder in eigenen Applikationen als „Self-Hosting“. Diese Bandbreite zusammen mit der Vielfalt an möglichen Bindings machen WCF mächtiger, aber auch komplexer und komplizierter als JAX-WS.

Da JAX-WS sich hauptsächlich mit SOAP/HTTP Webservices beschäftigt, begründet, dass sowohl der „Contract-First“ und „Code-First“ Ansatz zur Entwicklung von Diensten hier unterstützt werden. Dem reinen „Contract-First“ Ansatz, also das Ausgehen von einem WSDL-Dokument, wird in WCF wenig Beachtung geschenkt. Möglichkeiten existieren zwar über Commandline-Tools, eine Generierung aus dem WSDL-Dokument anzustoßen, diese sind aber nicht in die Entwicklungsumgebung integriert und gelten hier als alternativer Ansatz.

Sowohl in WCF als auch JAX-WS können Proxies für den Anwender generiert werden. Proxies vereinfachen die Kommunikation mit einem Service und abstrahieren entfernte Aufrufe. Die API und Benutzung dieser Proxies ist für beide Technologien absolut identisch.

Listing A.2: Proxy Nutzung in WCF

```
1 DemoServiceProxy proxy = new DemoServiceProxy();
2
3 int result = proxy.AddNumbers(1,2);
4
5 proxy.Close();
```

Listing A.3: Proxy Nutzung in JAX-WS

```
1 DemoServiceProxy service = new DemoServiceProxy();
2
3 DemoPort port = service.getDemoServiceSOAPPort();
```

[3]Oft als POJO - plain old java object bezeichnet

```
4
5 int result = port.AddNumbers(1,2);
```

Wie schon im vorigen Abschnitt erwähnt nutzt WCF wie JAX-WS Metadaten (in .NET Attribute und JAVA Annotations) um Dienste und ihre Methoden zu markieren. Dabei überdecken sich diese zwischen den Technologien großteils, wie A.4 und A.5 zeigen.

Listing A.4: Service Implementierung in JAX-WS

```
1  //Contract
2  @WebService(targetNamespace = "JAXDemo.Services", name="AddNumbers")
3  public interface DemoServiceIF extends Remote {
4
5          @WebMethod
6      public int AddNumbers(int number1, int number2) throws RemoteException;
7
8      @WebMethod
9      @OneWay
10     public void Save(int number) throws RemoteException;
11 }
12
13 //Implementation
14 @WebService(endpointInterface = "JAXDemo.Services.DemoServiceIF")
15 public class DemoServiceImpl {
16
17     public int AddNumbers(int number1, int number2){
18       return number1+number2;
19     }
20
21     public void Save(int number){
22         //doSomething
23     }
24
25 }
```

Listing A.5: Service Implementierung in WCF

```
1  //Contract
2  [ServiceContract(Namespace="WCFDemo.Service", Name="AddNumbers")]
3  public interface IDemoService
4  {
5      [OperationContract]
6      public int AddNumbers(int number1, int number2) throws RemoteException;
7
8      [OperationContract(IsOneWay=true)]
9      public void Save(int number);
10 }
11
12 //Implementation
13 public class DemoServiceImpl : WCFDemo.Services.IDemoService
14 {
15     public int AddNumbers(int number1, int number2)
16     {
17       return number1+number2;
18     }
19
20     public void Save(int number)
21     {
22         //doSomething
23     }
24
25 }
```

Diese Beispiele zeigen, dass die Umsetzung von Webservices sich in JAVA und .NET nicht besonders unterscheiden. Abschließend können die Eigenschaften der zwei Webservice APIs WCF und JAX-WS wie in Tabelle A.1 zusammengefasst werden.

Für das Standardszenario Webservices mit SOAP/HTTP stellen sich beide APIs als geeignet heraus. WCF ist ein umfassenders Kommunikationspaket und bietet mehrere unterschiedliche Protokolle und Bindings, wobei JAX-WS auf SOAP/HTTP baut. Um diese Aufgaben in JAVA zu lösen gibt es aber andere APIs, jedoch besticht die Möglichkeit von WCF verschiedenste Kommunikation gleich zu behandeln. Mit diesen Möglichkeiten steigt aber auch die Komplexität der API und die Konfiguration der

Merkmal	WCF	JAX-WS
Interoperabel	Ja	Ja
Protokollunterstützung	Vielseitig	hauptsächlich SOAP/HTTP
Komplexität	Hoch	Mittel
Code-First	Ja	Ja
Contract-First	Eingeschränkt	Ja
Integration in Entwicklungsumgebung	Sehr gut	Schlecht[4]
Hostingumgebung	App.Server, Windows Service, Selfhosting	App.Server

Tabelle A.1: Bewertung von WCF und JAX-WS

Bindings ist oft nicht trivial. Als kommerzielles Produkt ist WCF besser in die Entwicklungsumgebung integriet und Features wie Remote-Debugging von Services erleichtern die Entwicklung. Aus wissenschaftlicher Sicht ist JAX-WS wegen voller Unterstützung des Contract-First Ansatzes aber sicher näher am Servicegedanken.

A.2 Softwaredesign Überblick

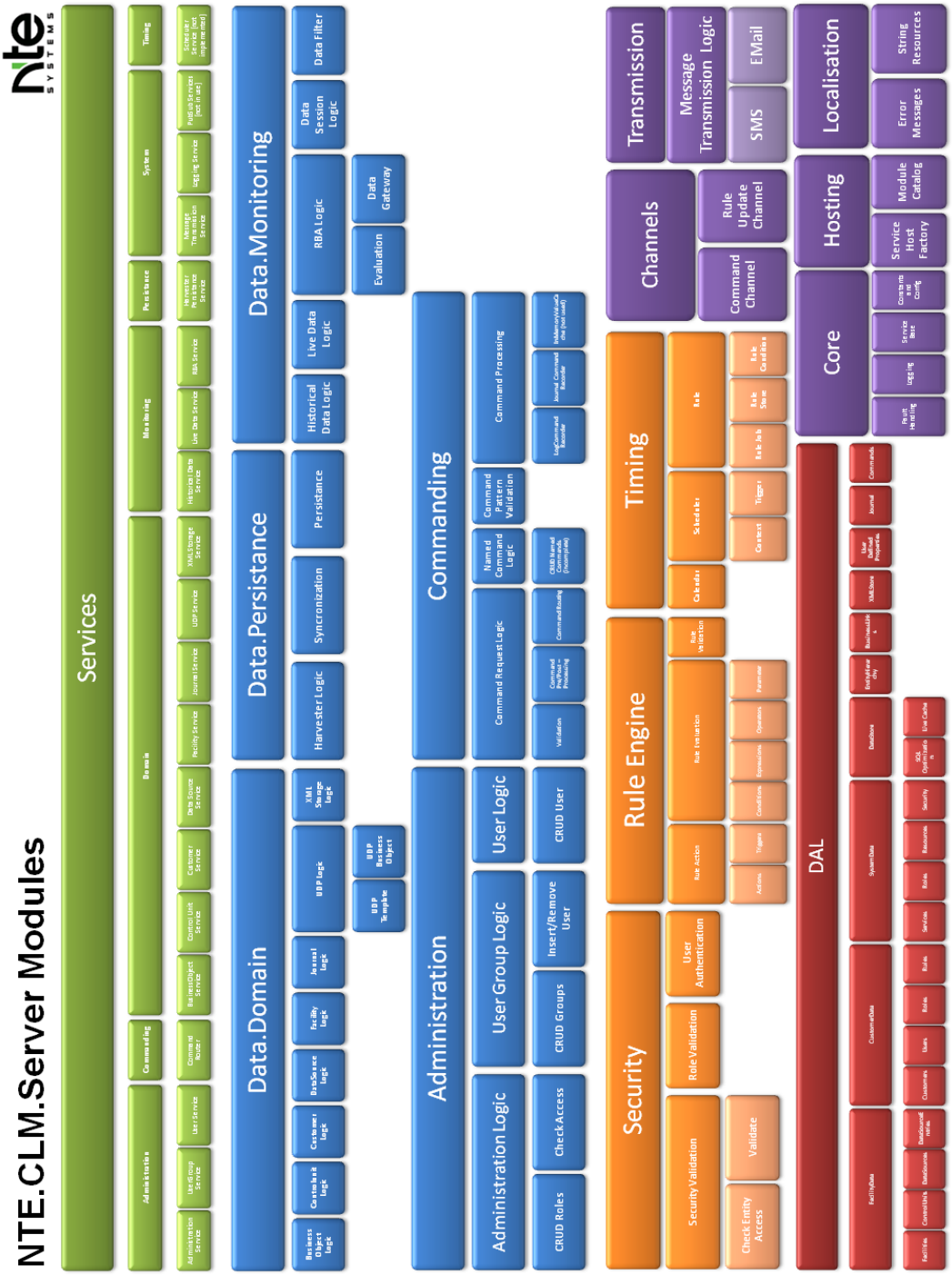

Abbildung A.1: Systemübersicht

A.3 Screenshots Frontends

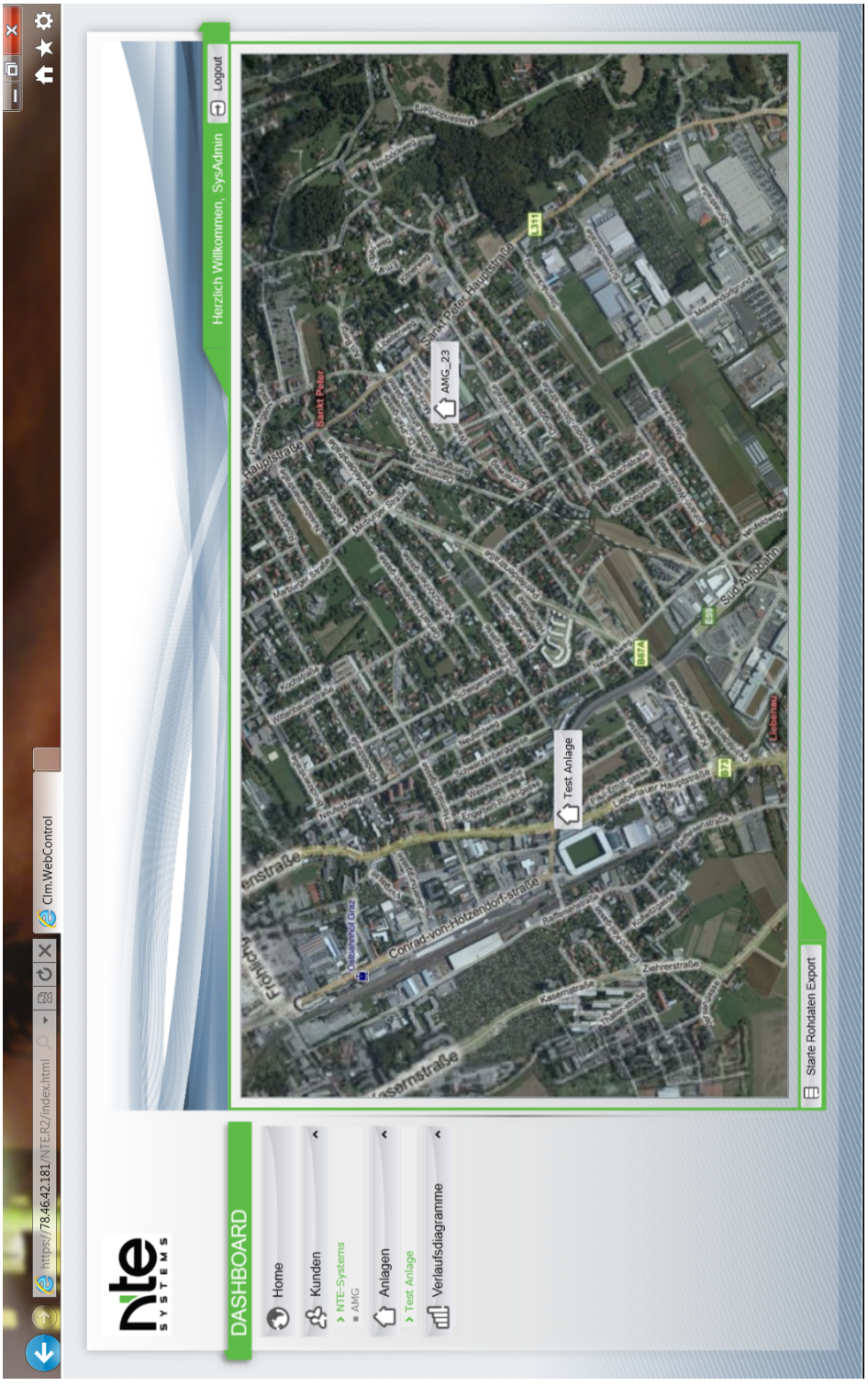

Abbildung A.2: CLM.WebControl, geolokalisierung von Anlagen

Abbildung A.3: CLM.WebControl, benutzerdefinierte Anlagenseite

Abbildung A.4: CLM.SystemDesigner, Editor für Anlagenseiten