Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3174

Service-orientierte Integration von proprietären Anwendungen der Digitalen Fabrik

Fabian Laux

Studiengang:	Informatik
Prüfer:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer:	Dipl.-Inf. Stefan Silcher Dipl.-Ing. Max Dinkelmann
begonnen am:	18. April 2011
beendet am:	18. Oktober 2011
CR-Klassifikation:	D.2.11, D.2.12, J.1

Kurzfassung

Die heutige Zeit der Globalisierung und der Zunahme individueller Kundenwünsche erfordert eine flexible Produktion von produzierenden Unternehmen. Um dieses Ziel zu erreichen werden immer öfters Werkzeuge eingesetzt, welche die Planung unterstützen und somit beschleunigen. Um den Planungsprozess zu optimieren wird versucht dieser Prozess zu modellieren, um ein Modell zu erhalten, das anschließend analysiert und verbessert werden kann.
Die Lernfabrik des Instituts für Industrielle Fertigung und Fabrikbetrieb der Universität Stuttgart (IFF) dient als Vorbild für Unternehmen. Sie besteht aus der physikalischen Fabrik, in der die neusten Methoden zur Produktoptimierung eingesetzt werden und der digitalen Fabrik.
Die digitale Fabrik setzt Werkzeuge ein, welche die Planungsphase von Fabriken und Produktionssystemen unterstützen. Diese Werkzeuge sind über Punkt-zu-Punkt Verbindungen miteinander verbunden, was mangelnde Flexibilität des gesamten Systems zur Folge hat.
Im Zuge dieser Arbeit soll eine neue Architektur erarbeitet werden, die bestimmte Werkzeuge der Layoutplanung integriert und zugleich eine erhöhte Flexibilität bietet. Außerdem soll die Layoutplanung in einem Workflow abgebildet werden.
Als Lösungsansatz wurde eine Architektur erstellt, die auf dem Prinzip der Serviceorientierten Architektur (SOA) beruht. In der SOA werden Services lose gekoppelt und über eine geeignete Infrastruktur miteinander verbunden. In dieser Arbeit wurde als Infrastruktur ein Enterprise Service Bus (ESB) verwendet. Zur weiteren Unterstützung der losen Kopplung und somit zur Erhöhung der Flexibilität des Gesamtsystems, wurden Message Queues eingesetzt. Durch die Verwendung von Services war es möglich einen Workflow zu modellieren, der die Layoutplanung abbildet.
Durch die eingesetzten Technologien wurden Werkzeuge der Layoutplanung integriert und die Flexibilität des Gesamtsystems erhöht. Außerdem konnte die Layoutplanung in einem Workflow modelliert werden, was eine Analyse des Planungsablaufs und somit auch eine Optimierung der Planung ermöglicht.
Durch die lose Kopplung der Systeme, was das Austauschen von Planungswerkzeugen erheblich vereinfacht, wurde die Flexibilität des Gesamtsystems erhöht. Der Lösungsansatz ist somit nicht nur auf die Lernfabrik des IFF beschränkt, sondern kann auch in produzierenden Unternehmen eingesetzt werden, da ein Großteil der Unternehmen ebenfalls planungsunterstützende Werkzeuge einsetzt, die geeignet miteinander verbunden werden müssen um die Planung bestmöglich zu unterstützen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CSS	Cascading Style Sheets
DPE	Delmia Process Engineer
ESB	Enterprise Service Bus
FIFO	First in First out
GUID	Globally Unique Identifier
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IFF	Institut für Industrielle Fertigung und Fabrikbetrieb
IIS	Internet Information Services
JBI	Java Business Integration
JDBC	Java Database Connectivity
JMS	Java Message Service
LDAP	Lightweight Directory Access Protocol
MEX	Web-Service-Metadata Exchange Protocol
MSMQ	Microsoft Message Queue
NMR	Normalized Message Router
PDA	Personal Digital Assistant
PPR	Product Process Resource Navigator:
QOS	Quality of Services
RPC	Remote Procedure Call
SAO	Server Activated Objects
SOA	Service Oriented Architecture
SQL	Structured Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VBA	Visual Basic for Applications
WAS	Windows Activation Service
WSDL	Web Service Description Language
WSFL	Web Service Flow Language
XML	Extensible Markup Language
XSD	XML Schema

1. Einleitung

Eine digitale Fabrik beinhaltet Modelle, Methoden und Werkzeuge, die dazu dienen ein Abbild einer realen Fabrik darzustellen. Dazu gehören softwaregestützte Systeme für Planung und Analyse des realen Fabrikbetriebs. Die reale Fabrik setzt sich unter anderem aus Produktionsmaschinen, Gebäuden und den Mitarbeitern des Unternehmens zusammen [Gru09].
Die Lernfabrik des Institutes für Industrielle Fertigung und Fabrikbetrieb (IFF) beinhaltet sowohl eine digitale Fabrik, die als Lerninsel bezeichnet wird, als auch eine reale Fabrik, welche die neuesten Methoden zur Produktoptimierung einsetzt. Diese wird als physikalische Modellfabrik bezeichnet.
Die Lernfabrik dient als Vorbild für Unternehmen, die ihre Firmenstruktur an den neusten Forschungsansätzen ausrichten wollen. Um das Potential der Lernfabrik aufzuzeigen wird ein Schreibtischset gefertigt. Dabei kommt das Zusammenspiel zwischen Lerninsel und physikalischer Modellfabrik richtig zur Geltung. In der Lerninsel wird die Fertigung geplant und simuliert, um dann später mit Hilfe der gewonnen Daten die Produktionsstätten optimal an ihren Einsatz anzupassen [ler].

1.1. Motivation

Durch die zunehmende Globalisierung der Wirtschaft und das Wachsen individueller Kundenwünsche ist es heutzutage für Unternehmen wichtiger den je die Produktion rasch umzustellen. Ein Ziel, das mit Hilfe einer digitalen Fabrik besser erreicht werden kann. Mit Hilfe von verschiedenen Planungs- und Simulationswerkzeugen der digitalen Fabrik können veränderte Anforderungen, ohne großen Aufwand, direkt in die Produktionsstrukturen abgebildet werden. Dies erhöht die Flexibilität und zudem werden Kosten eingespart [Küh06] [Gru09].
Die Vernetzung von realer und digitaler Fabrik und dessen Werkzeugen führt zur Vermeidung von Datenredundanz, erhöht die Aktualität der Daten und es ist sichergestellt, dass die Planung realistische Szenarien verwendet. Unter *den Daten* versteht man in diesem Zusammenhang die Anzahl der Maschinen, dessen Durchlaufzeiten, die Anzahl der Mitarbeiter, die Größe der Hallen, im Prinzip alles was als Faktor in die Fertigung eingeht [BGW11].
Durch die komplette Digitalisierung der Planungs-, Simulations- und Fertigungsschritte ist es möglich die Geschäftsprozesse zu erfassen, später zu analysieren und bei Bedarf verbessern und umzustellen. Diese Geschäftsprozesse bilden die Wertschöpfungskette eines Unternehmens. Wenn die Lieferanten und Kunden in diese Prozesse noch miteinbezogen werden,

dann hat dies sogar noch Auswirkungen auf die logistische Planung des Unternehmens [SW04].

1.2. Problemstellung

Die Vernetzung einer digitalen Fabrik basiert bisher meistens auf Punkt-zu-Punkt-Verbindungen der eingesetzten Werkzeuge oder sie wurden durch einen Bus miteinander verbunden. Doch dies schränkte die Flexibilität der kompletten digitalen Fabrik erheblich ein. So auch bei der Lernfabrik, da es nicht möglich ist einzelne Werkzeuge auszutauschen ohne dabei die komplette Infrastruktur des Systems ändern zu müssen. Ein weiteres Problem sind die unterschiedlichen Datenformate der jeweiligen Werkzeuge. Da die Tools jeweils miteinander verbunden sind, müssen die Daten für jedes Werkzeug angepasst werden, was zu einem erheblichen Aufwand führt, der sich bei Hinzunahme oder Änderungen an einem System multipliziert.
Ein weiteres Problem bei der momentanen Implementierung ist die fehlende Unterstützung von Geschäftsprozessen. Bisher ist es nicht möglich diese abzubilden. Somit besteht auch keine Möglichkeit diese zu optimieren, was dazu führt, dass eventuell vorhandene Potentiale, wie Zeit- oder Kosteneinsparungen, nicht ausgeschöpft werden können.

1.3. Zielsetzung

Ziel dieser Arbeit ist es ausgewählte Systeme des Planungsvorganges, angefangen von Datenhaltung bis hin zur eigentlichen Planung, als Services zur Verfügung zu stellen. Diese Services sollen dann mit einem ESB verbunden werden. Als weitere Unterstützung zur losen Kopplung der Systeme sollen Warteschlangen eingesetzt werden, auf denen die Nachrichten gespeichert werden, falls ein System kurzfristig ausfällt. Nachdem das System wieder verfügbar ist, soll der reguläre Betrieb fortgesetzt werden. Ein weiteres Ziel dieser Arbeit ist es den Planungsablauf als Workflow abzubilden.

1.4. Vorgehensweise

In Kapitel 2 werden die fachlichen Grundlagen und die Anforderungen für das Erreichen der Ziele erörtert. Des weiteren werden die dafür notwendigen theoretischen Grundlagen und die darauf basierenden eingesetzten Technologien besprochen.
Das Kapitel 3 stellt die verwendete Architektur vor. Hier werden nicht nur die vorhandenen Werkzeuge, sondern auch die Schnittstellen zu diesen und dessen Erweiterungen besprochen. Am Ende des Kapitels wird das Augenmerk auf die Implementierung der einzelnen Systeme und die abgebildeten Geschäftsprozesse gelegt.
In Kapitel 4 wird eine Bewertung aufgestellt. Dies vergleicht die gestellten Anforderungen mit den gewünschten Zielen. Außerdem werden Probleme, die während der Umsetzung

aufgetaucht sind, besprochen.
In Kapitel 5 befindet sich eine kurze Zusammenfassung der Arbeit. Im Anschluss daran werden mögliche Erweiterungen diskutiert.

2. Grundlagen

In diesem Kapitel werden die Grundlagen für das Verständnis dieser Arbeit erläutert und definiert.
Der erste Teil beschäftigt sich mit der digitalen Fabrik und dem Begriff der Produktionsplanung.
Im zweiten Teil werden die theoretischen Grundlagen besprochen, die in für die Ausarbeitung nötig sind. Als Grundbaustein dient die *Service Oriented Architecture* (SOA). Um diese implementieren zu können sind weitere Ansätze wie der *Enterprise Service Bus* (ESB) oder die *Microsoft Message Queue* (MSMQ) notwendig.
Im dritten Teil werden die eingesetzten Systeme und Technologien besprochen, die für die Implementierung nötig sind.

2.1. Fachlicher Hintergrund

In diesem Abschnitt wird der Begriff der digitalen Fabrik erörtert und ein Zusammenhang zur Fabrik- und Produktionsplanung hergestellt.

2.1.1. Digitale Fabrik

Als Digitale Fabrik bezeichnet man ein komplexes Netzwerk von Modellen, Methoden und speziellen Werkzeugen, die alle miteinander verbunden sind und auf die selben Daten zugreifen [Gru09]. Es wird ein Abbild einer realen Fabrik erzeugt, woraus sich die Möglichkeit ergibt, diese durch Planungsdaten zu simulieren. Dies birgt einige Vorteile mit sich, wie zum Beispiel eine erhöhte Planungsgeschwindigkeit, eine verbesserte Qualität der Planung und steigende Flexibilität im Bezug auf Produktionsänderungen.
Im Zuge der Globalisierung ergeben sich zunehmend turbulente Märkte, welche neue Anforderungen an das Fabrik- und Produktionssystem stellen [Gru09]:

Flexibilität: In diesem Kontext versteht man unter Flexibilität die Veränderungsfähigkeit im Sinne von Umrüstbarkeit, Rekonfigurierbarkeit und Kapazitätserhöhung des Materialflusses. Diese Änderungsfaktoren beruhen auf den Grundstrukturen der Fabrikgestaltung und sind daher kurzfristig aktivierbar.

Wandlungsfähigkeit: Hierunter versteht man langfristige Änderungen. Wie zum Beispiel das Ändern von Gebäuden und der Gebäudeinfrastruktur.

Die Flexibilität kann in vielen Bereichen verändert und erhöht werden. So kann versucht werden verschiedene Produktionsmaschinen flexibler zu gestalten um sie vielseitiger einsetzen zu können, oder durch die Änderung des Produktionsprozesses, um den Materialfluss zu erhöhen .
Doch diese Arbeit beschäftigt sich mit der Flexibilität der Informations- und Kommunikationssysteme. Durch das Schaffen variabler Kommunikationssysteme wird versucht alle eingesetzten Planungswerkzeuge effizient miteinander zu verbinden.

2.1.2. Fabrik- und Produktionsplanung

Um Synergieeffekte zu erreichen werden immer häufiger Produktentwicklung und Produktionsplanung weitgehend parallel durchgeführt. Durch den Parallelismus wird versucht die Einführungszeit für ein Produkt zu reduzieren und eventuelle Optimierungspotentiale bereits vor Beginn der Produktion zu erkennen [Küho6].
Zu den Aufgaben der Produktionsplanung gehören unter anderem die Planung der Produktionsstätten, die Überwachung der dazugehörigen Systeme, sowie die Realisierung der Produktionsanlagen. Damit gehört die Produktionsplanung zum wesentlichen Anwendungsgebiet der digitalen Fabrik und umfasst folgende Teilgebiete:

- Technologieplanung
- Prozessplanung
- Layoutplanung
- Arbeitsplatzgestaltung

Um die *Layoutplanung* effizient durchführen zu können wird der Planungstisch eingesetzt, mit dessen Hilfe 3D Modelle erstellt werden können. Der Planungstisch wird in Abschnitt 3.3.3 genauer beschrieben.
Der Umfang der Planung hängt davon ab wie stark ein Geschäftsprozess verändert werden soll. So können einzelne Arbeitsplätze umgestaltet oder komplette Produktionsstätten neu konzipiert und umgesetzt werden. Dabei wird nicht nur auf die technischen Umsetzung und die wirtschaftliche Komponente geachtet, sondern auch auf eine ergonomische Gestaltung von Arbeitsplätzen. In den jeweiligen Planungsschritten entstehen unterschiedliche Modelle, die simuliert und anschließend bewertet werden können [Küho6].

2.2. Theoretische Grundlagen

In diesem Abschnitt werden alle theoretischen Grundlagen zum Verständnis dieser Arbeit vorgestellt.

2.2.1. Service Oriented Architectures

SOA ist keine Technologie, sondern ein Konzept, welches in den letzten Jahren deutlich an Bedeutung gewonnen hat. Vor allem weil der Fokus nicht mehr auf der Mensch-zu-Mensch Kommunikation liegt, sondern viel mehr auf der Kommunikation zwischen verschiedenen Anwendungen. Der Grundgedanke dieses Konzeptes liegt darin, verschiedene Services miteinander zu verbinden, die dann zusammen einen Geschäftsprozess darstellen. Dabei stellt jeder Service für sich wieder einen Prozess dar [Jos09].
Im Folgenden werden aus den verschiedenen Definitionen [Bur07],[Mat08] [Mel08] und [FZ09] Gemeinsamkeiten erarbeitet, welche in dieser Arbeit verwendet werden.

Service: Ein Service wird erstellt um ihn einer übergeordneten Instanz zur Verfügung zu stellen. Services werden normalerweise in einer maschinenlesbaren Sprache beschrieben, gängig dafür ist Extensible Markup Language (XML). Es werden allerdings nur die Schnittstellen beschrieben, wie der Service implementiert ist bleibt verborgen.

Service-Provider: Dieser ist der Anbieter des Dienstes und wird deshalb auch Dienstanbieter genannt.

Service-Consumer: Dieser nimmt den angeboten Dienst in Anspruch. Da er den Dienst nutzt, wird auch von Dienstnutzer gesprochen.

Service-Kapselung: Unter diesem Stichwort versteht man die Unabhängigkeit zweier Systeme.

Wiederverwendbarkeit: Durch Wiederverwendbarkeit soll erreicht werden, dass keine redundanten Implementierungen notwendig sind.

Lose Kopplung: Dienste werden nicht fest miteinander verbunden. Erst bei Bedarf wird nach ihnen gesucht und diese dann eingebunden. Da das Suchen und Binden eines Services erst zur Laufzeit geschieht, wird von loser Kopplung gesprochen. Durch die lose Kopplung wird die Wiederverwendbarkeit erhöht und ermöglicht.

Dynamisches Binden: Um einen Service erst zur Laufzeit binden zu können muss sichergestellt sein, dass es ausreichend Informationen zur Suche und später dann auch zum Binden des Services gibt.

Service-Registry: Oder auch als Dienstverzeichnis bekannt. Darin werden alle Services registriert, die zur Verfügung stehen.

Skalierbarkeit: Durch genau definierte Schnittstellen ist es möglich bei Bedarf weitere Systeme einzusetzen. Bei hoher Auslastung können weitere Komponenten eingesetzt werden um die Last gleichmäßig auf alle Systeme verteilen zu können. Dies führt zu schnelleren Antwortzeiten.

Verwendung von Standards: Um komplexe Systeme zu implementieren oder Services öffentlich zu machen wird auf definierte Standards gesetzt. Dabei wird ein Paradigma der Softwareentwicklung verfolgt -> Trennung zwischen Schnittstelle und Implementierung.

Nichtfunktionale Eigenschaften: Diese beschreiben nicht die Funktionalität des Services, sondern qualitative Eigenschaften. Beispielsweise die Kosten. Anhand dieser nichtfunktionalen Eigenschaften lässt sich die Servicequalität bestimmen. Bei Aufruf des Services verpflichtet sich der Service-Provider diese gegenüber dem Service-Consumer einzuhalten. Dies ist wie ein abgeschlossener Vertrag, wird dieser nicht eingehalten, so können dem Service-Provider rechtliche Konsequenzen drohen.

Das SOA-Dreieck

Die Verwendung und der Aufruf von Web Services beruhen auf dem so genannten SOA-Dreieck.

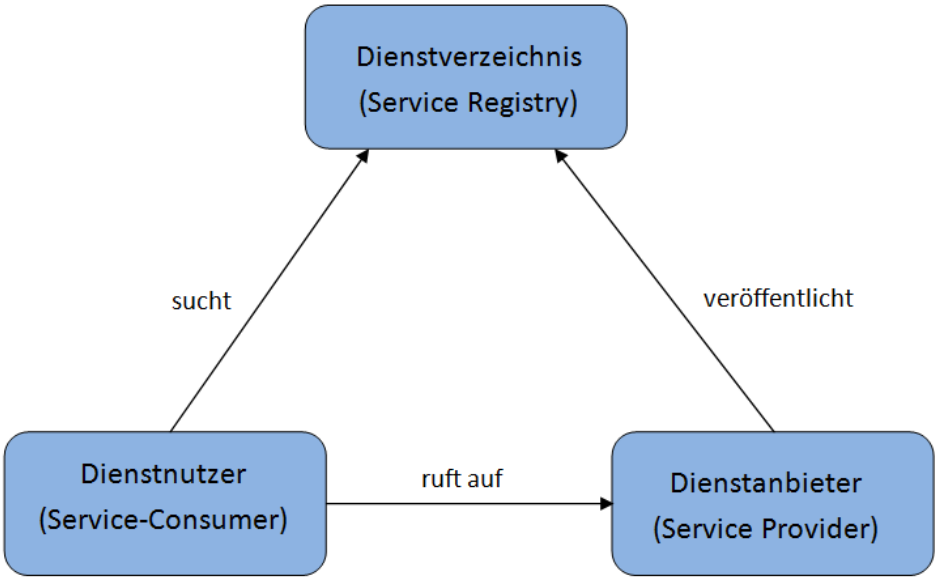

Abbildung 2.1.: Das SOA-Dreieck [CG]

Abbildung 2.1 stellt den im folgenden beschriebenen Ablauf eines Web-Services-Aufrufs bildlich dar:

- Der Dienstanbieter veröffentlicht den Service im Dienstverzeichnis.
- Der Dienstnutzer sucht im Dienstverzeichnis nach einem geeigneten Service.
- Wird ein Service gefunden, so wird er mit Hilfe der Servicebeschreibung gebunden und aufgerufen.

2.2.2. Business Process Execution Language

Die Modellierung von Geschäftsprozessen wird heute in Unternehmen immer öfters angewendet, da dadurch ein Mehrwert für das Unternehmen entsteht. So können bereits vorhandene Services in neue Prozesse integriert werden und müssen nicht neu erstellt werden [Melo8].
Die Business Process Execution Language (BPEL) ist ein Sprachstandard zum Definieren von Geschäftsprozessen, der aus der *Web Service Flow Language (WSFL)* von IBM und Microsofts *XLANG* hervorging [Joso9].
Das Erstellen eines neuen Services aus verschiedenen, bereits vorhandenen Services, nennt man *orchestrieren*. Üblicherweise werden solche Services als Web Services (WS) angeboten, deswegen hat sich der Name *BPEL4WS* etabliert, bevor er dann in *WS-BPEL* umbenannt wurde [Joso9].
BPEL ist eine XML-basierte Programmiersprache, die wie folgt aufgebaut ist:

Variables: Hier werden Prozessvariablen definiert, die global im ganzen Programm verwendet werden können.

Sequence: Definiert eine Ablaufreihenfolge von Aktivitäten.

Flow: Entspricht einer Sequence ohne festgelegte Reihenfolge. Jede Aktivität kann zur jeder Zeit ausgelöst werden.

Receive: Empfängt eine Nachricht und fährt dann fort. Entspricht einem externen Aufruf des Services.

Assign: Wird benutzt um Daten zu mappen. Beispielsweise von einer Variable in die andere.

Invoke: Dient als Aufruf anderer Services.

Reply: Legt das Prozessende fest. Hier wird definiert welche Daten an die aufrufende Instanz zurückgegeben werden.

Da dieser Aufbau allerdings nicht gerade überschaubar und komfortabel ist, gibt es reichlich Tools, die eine handliche Oberfläche besitzen. Diese Oberfläche ist intuitiv und stellt die einzelnen Bausteine als grafische Symbole zur Verfügung. Mit Hilfe dieser Symbole lässt sich ein Prozess auf einfacherem Wege erstellen [Melo8] [Joso9].

2.2.3. Enterprise Service Bus

Der Enterprise Service Bus stellt eine Infrastruktur für Service Oriented Architectures zur Verfügung. Es gibt verschiedene Infrastrukturen um den Einsatz einer SOA zu ermöglichen. Voraussetzung ist, dass es eine Middleware ist, die alle SOA-Prinzipien unterstützt.
Als Middleware bezeichnet man eine Softwareschicht zwischen zwei kommunizierenden Anwendungen. Dabei hat die Middleware unter anderem die Funktion zwischen heterogenen Datenformaten zu vermitteln [Kaio4].
Diese Infrastruktur ist die Grundvoraussetzung um Services systemübergreifend aufrufen

zu können [Jos09]. Früher gab es Kommunikationsnetze mit verschiedenen Austauschprotokollen, diese wurde durch den Bus abgelöst, der die Nachrichten in einem unabhängigen Datenformat von Sender zum Empfänger transportiert und somit eine lose Kopplung der Systeme gewährleistet [Bib08]. Der ESB wird oft als Backbone einer SOA, mit intelligenten Routingfähigkeiten bezeichnet. Durch das Routen wird das dynamische Binden von Services gewährleistet. Erst zur Laufzeit wird der richtige Service gesucht, gefunden und aufgerufen [Mel08]. Jedoch gibt es auch hier verschiedene Ansätze den Begriff einzugrenzen, deswegen werden im Folgenden verschiedene Definitionen abgeleitet [Bib08][Mel08].

Transformation: Da verschiedene Systeme mit verschiedenen Datenformaten über ihn kommunizieren können müssen, muss eine Transformation der Daten auf ihm stattfinden. Dabei gibt es verschiedene Stufen der Transformation. Beispielsweise muss nur von Celsius in Fahrenheit umgerechnet werden, was leicht zu realisieren ist. Ein anderes Beispiel ist die Zuordnung von Adressen. Einmal ist die Hausnummer in die Adresse miteinbezogen und im anderen Datenformat gibt es zwei verschiedene Bereiche, Adresse und Hausnummer. Aber es gibt auch deutlich komplexere Aufgaben bei der Transformation, wie zum Beispiel das Mappen unterschiedlicher Datenbankschemen.

Protokollunabhängigkeit: Nicht nur mit verschiedenen Datenformaten muss umgegangen werden, sondern auch mit verschiedenen Arten der Kommunikation. So darf es keinen Unterschied darstellen wie mit den verschiedenen Services kommuniziert wird. Beispielsweise ist der eine über das Hypertext Transfer Protocol (HTTP) erreichbar und der andere über einen Remote Procedure Call (RPC).

Intelligentes Routing: Damit Services lose gekoppelt sind, ist es wichtig, dass sie die Endpunkte der anderen Systeme nicht kenne müssen. Sie schicken ihre Nachrichten an die Routingkomponente des ESB, die das Routen übernimmt.
Dies kann mit einem normalen Netzwerkswitch verglichen werden. Dieser erkennt anhand der Nachricht an welchen Empfänger diese geschickt werden soll.

2.2.4. Message Queuing

Unter einer Message Queue, zu deutsch Warteschlange, versteht man eine Datenstruktur zum Verwalten von Nachrichten, die nach dem First In First Out (FIFO) Prinzip arbeitet. Das heißt es wird ein Puffer verwendet, der von der einen Seite gefüllt und von der anderen Seite ausgelesen wird. Dabei bleibt jedes Objekt, nach dem es auf die Warteschlange gelegt wurde, so lange in ihr, bis es ausgelesen wird.
In den meisten Fällen wird keine Prioritätswarteschlange verwendet, das heißt es ist nicht möglich ein Element, das später auf die Warteschlange gelegt wurde, früher wieder auszulesen.
Message Queues werden in der Informatik häufig verwendet, wenn zwei Anwendungen asynchron kommunizieren.
Der Vorteil dieser Technik ist, dass die aufgerufene Anwendung während des Aufrufs nicht verfügbar sein muss, sondern sich die Nachricht auch später abholen kann [MKG09].

2.3. Eingesetzte Technologien

In diesem Abschnitt werden die eingesetzten Technologien dieser Arbeit vorgestellt.

2.3.1. Open ESB

Als Enterprise Service Bus für diese Arbeit wurde der Open ESB gewählt, da er den Definitionen von 2.2.3 entspricht, bereits einen Application Server beinhaltet und damit eine geeignete Plattform für die Implementierung einer SOA-Landschaft bietet. Außerdem trugen folgende Argumente dazu bei sich für den Open ESB zu entscheiden [BHR07]:

- Die Software steht als Open Source und damit kostenfrei zur Verfügung.
- Gute Dokumentation der Software.
- Eine große und aktive Community.
- Stellt viele Funktionen bereit.
- Guter Support.

Die Entwicklung basiert auf dem Application Server und stellt zusätzlich das Netbeans Integrated Development Environment (IDE) zur Verfügung. Netbeans IDE ist eine Entwicklungsumgebung, die komplett in Java geschrieben ist [BHR07]. Mit Hilfe seiner Java Business Integration (JBI) Anbindung, siehe Abschnitt 2.3.8, stellt er folgende Binding Components (BC) zur Verfügung [gla]:

Database-BC: Ermöglicht den Zugriff auf Datenbanken mit Hilfe des Java Database Connectivity (JDBC) Treibers. Die Implementierung entspricht der JBI Specification 1.0.

File-BC: Ermöglicht den Zugriff auf das Dateisystem im JBI-Umfeld. Außerdem können JBI-Nachrichten normalisiert und in Dateien eines angegebenen Pfades geschrieben werden.

FTP-BC: Ermöglicht das Senden und Empfangen von Nachrichten über verschiedene Protokolle. Dabei werden die Nachrichten normalisiert damit diese vom JBI-Umfeld losgelöst werden. Die Implementierung entspricht ebenfalls der JBI Specification 1.0.

HTTP-BC: In diesem Binding werden SOAP-Nachrichten über das HTTP-Protokoll verschickt. Doch es dient nicht nur dazu Nachrichten zu verschicken, sondern auch als Endpunkt um von anderen Anwendungen auf den ESB zuzugreifen.

JMS-BC: Java Message Service ermöglicht den Anschluss an eine Message Oriented Middleware (MOM), welche ihre Schnittstellen in der Web Service Description Language (WSDL) beschreibt.

LDAP-BC: Mit Lightweight Directory Access Protocol (LDAP) Binding Component ist es möglich auf ein LDAP-Verzeichnis auf einem LDAP-Server zuzugreifen.
LDAP ist ein Kommunikationsprotokoll zwischen Active Directory-Clients und -Servern. Dieser Protokoll legt fest wie der Zugriff auf die Active Directory-Server erfolgen darf und welche Daten benutzt werden dürfen. Über LDAP werden Objekte gelesen und gespeichert [KT05].

MSMQ-BC: Bietet die Möglichkeit auf eine Microsoft Message Queue zuzugreifen.

In dieser Arbeit wurde sich darauf beschränkt das MSMQ-BC zu benutzen. Da alle in Kapitel 3 vorgestellten Systeme über eine Microsoft Message Queue verfügen.

2.3.2. .NET

.Net ist eine Entwicklungsplattform, die von Microsoft entwickelt wurde. Die wichtigsten Komponenten sind die Laufzeitumgebung mit Namen Common Language Runtime (CLR) und die .Net-Framework Klassenbibliothek. Die CLR basiert, wie die JAVA Runtime auch, auf einer virtuellen Maschine. Das .Net-Framework stellt neben einer verwalteten Ausführungsumgebung, das heißt der Code wird in einen Zwischencode, der sogenannten Common Intermediate Language (CIL), kompiliert und erst zur Laufzeit in einen Maschinencode kompiliert, eine vereinfachte Entwicklungsumgebung bereit (siehe Abbildung 2.2) [Sta04]. Es ist möglich .Net in vielen Programmiersprachen einzubinden. Unter anderem in Visual Basic oder wie in dieser Arbeit in C# [rem].

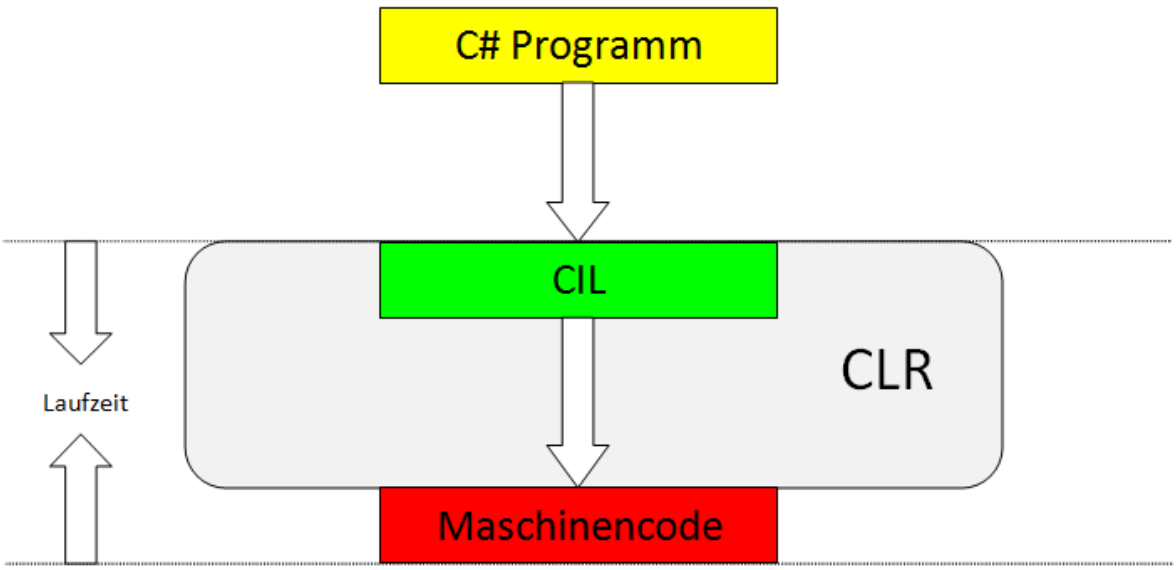

Abbildung 2.2.: Übersicht Common Language Runtime [Sta04]

.Net-Remoting

In den Abschnitten 3.3.6 und 3.3.7 wird .Net-Remoting Framework benötigt, deswegen wird dieses hier genauer erläutert. .Net-Remoting ist der Nachfolger von Component Object Model (COM) (siehe Abschnitt 2.3.5). Das Framework dient zur Kommunikation zwischen Objekten und unterstützt verschiedene Übertragungsprotokolle.
Mit Remoting ist es möglich Objekte aus einem entfernten Kontext aufzurufen. Dabei ist es egal, ob die verschiedenen Objekte auf dem gleichen oder einem entfernten Rechner liegen. Man unterscheidet zwischen folgenden Arten von Services [Buro7]:

Remoting Services Es wird auf entfernte Objekte zugegriffen.

Enterprise Services Stellt Zugriffsklassen für die COM+-Infrastruktur zur Verfügung. Die COM+-Architektur besteht der früheren COM-Struktur und zusätzlichen Erweiterungen. Enterprise Services zusammen mit COM+ stellen im Grunde einen Application Server dar.

Da in den Abschnitten 3.3.6 und 3.3.7 ausschließlich Remoting Services verwendet werden, werden im Folgenden einige Begriffe genauer erläutert [Buro7].

Marshaling: Dabei wird ein Programmcodeaufruf verschickt, der es ermöglicht Objekte gemeinsam zu nutzen, obwohl sie keinen gemeinsamen Speicher besitzen.

Proxies: Durch Proxies wird die Kommunikation von Remoteobjekten ermöglicht. Das heißt das Erzeugen eines Proxyobjekts ist wie eine lokales Abbild des Remoteobjekt und leitet lokale Aufrufe an das Remoteobjekt weiter.

Messages: Messages sind Prozeduraufrufe, die zwischen den Endpunkten verschickt werden.

Channels: Über Channels werden die einzelnen Nachrichten transportiert. Die Channels übernehmen das Routing, sowohl auf Client- als auch auf der Serverseite.

Formatters: Sind für das Ver- und Entschlüsseln verantwortlich. Es sind sowohl SOAP- als auch Binary-Formatierungen möglich.

Single Call: Diese Objekte bearbeiten genau eine Anfrage eines Clients. Dabei können sie Statusinformationen speichern oder weiterleiten.

Singleton: Diese Objekte können von mehreren Clients angesprochen werden, dabei greifen alle Clients auf eine Datenbasis zurück. Man spricht auch von *Globalen Variablen auf Objektebene*.

Client Activated Objects (CAO): Das sind serverseitige Objekte, die vom Client aktiviert werden und genau dieser bestimmt auch über die Lebensdauer des Objekts.

Server Activated Objects (SAO): Vom Server aktivierte Objekte, dessen Lebensdauer ebenfalls vom Server bestimmt wird.

Für die Kommunikation zwischen den Services in Abschnitt 3.3.6 und 3.3.7 werden Singleton-Calls von Client Activated Objects verwendet.
Die Übergabe von Parametern bei .Net-Remoting wird unterschieden in [Bur07]:

- Parameter in Methoden
- Rückgabewert von Methoden
- Wert eines Properties oder Feldzugriffs

Dabei wird unterschieden in:

MarshalByValue: Das bedeutet, es wird eine Kopie des Objektes erstellt. Nur diese Kopie wird dann von einem Endpunkt zum anderen übertragen.

MarshalByReference: Es wird nur eine Referenz auf das Objekt verschickt. Erst beim Empfänger wird diese Referenz in ein Proxyobjekt umgewandelt und kann somit wieder verwendet werden. Dabei werden sämtliche Aufrufe des Proxyobjekt an die Referenz und somit dem ursprünglichen Objekt weitergeleitet [Bur07].

Vergleicht man das .Net-Remoting Framework mit Web Services so lassen sich folgende grundsätzliche Unterschiede feststellen:

- Das .Net-Remoting Framework bietet im Gegensatz zu Web Service ein Objektmodell an.
- In .Net-Remoting wird ein binäres Format benutzt, mit dessen Hilfe ganze Objekte zwischen zwei verschiedenen Anwendungen übermittelt werden können. Web Services hingegen unterstützen nur das Übertragen von XML.
- Um das .Net-Remoting verwenden zu können muss der Entwickler auch die Verwaltung der Objekte auf der Serverseite programmieren. Dies ist bei der Benutzung von Web-Services nicht notwendig [PP04].

2.3.3. Windows Communication Foundation

Die Windows Communication Foundation ist ein Framework zum Erstellen von Services, beruhend auf dem .Net-Framework. Als Hauptentwurfsziel der Windows Communication Foundation (WCF) gilt die Flexibilität bei der Kommunikation zwischen Client und Server. Es wird strikt zwischen Implementierung und Kommunikation getrennt. So kann man ohne Änderung der Implementierung das Kommunikationsprotokoll ändern, in dem man die Konfiguration des Services ändert. Durch Anwendung von Standards ist es problemlos möglich anbieterübergreifend zu kommunizieren. Folgende Definitionen werden in dieser Arbeit verwendet [KH09].

Contracts: Der *ServiceContract* und der *OperationContract* beschreiben die Funktionalität des Services. Der *DataContract* beschreibt die Parameter, die der Service empfängt oder zurückgibt. Der *FaultContract* dient zum Beschreiben eventuell auftretender Fehler und dessen Behandlung. Diese Contracts zusammen bilden das Interface des Services.

Endpunkte: Die Endpunkte definieren wie ein Service verfügbar gemacht wird. Jeder Endpunkt besteht aus drei Teilen, welche auch als ABC der Endpunkte bekannt sind:

- Einer **A**dresse, der Uniform Resource Identifier (URI), diese beschreibt wo die Nachricht für den Service hin geschickt werden soll.
- Einer **B**indung, die Informationen über die Kommunikation beinhaltet.
- Der **C**ontract, gibt Aufschluss über die Funktionalität des Services.

Metadaten: Über Metadaten wird beschrieben welche Merkmale ein Service besitzt und wie mit den Endpunkten kommuniziert wird. Das Veröffentlichen der Metadaten eines Services geschieht über spezielle *Metadaten-Endpunkte*. Diese werden über das HTTP-GET-Protokoll oder über ein spezielles Web Service Protokoll, das WS-Metadata Exchange Protocol (MEX), zur Verfügung gestellt. Ein Service kann folgende Metadaten veröffentlichen:

- Methoden des Services und dessen Beschreibung. Diese werden in WSDL veröffentlicht.
- Aller Parameter, die der Service verwendet.
- Nichtfunktionale Eigenschaften des Services.

Laufzeitverhalten: Mit Hilfe sogenannten *Behaviors* ist es möglich die Nachrichtenübermittlung zwischen Client und Server während der Laufzeit zu beeinflussen. Diese werden allerdings, im Gegensatz zu den Metadaten, lokal für einen Client oder Service festgelegt und nicht ausgetauscht. Dabei lassen sich die Behaviors in folgende Gruppen einteilen:

- EndpointBehaviors legen Verhaltensweisen fest, die sich auf die Endpunkte auswirken.
- ServiceBehaviors steuern das interne Ausführungsverhalten einer Service-Implementierung.
- OperationBehaviors beeinflussen die Verhaltensweisen einzelner Operationen.
- CustomBehaviors sind benutzerdefinierte Verhaltensweisen, die den kompletten Service, die Endpunkte oder einzelne Operationen beeinflussen.

Messaging: Hierzu gehören Kanäle und Encoder. Über die Kanäle werden die Nachrichten verschickt. Mit Hilfe des Encoders werden die Objekte serialisiert und deserialisiert.

Hosting: Jeder Service ist ein eigenständiges Programm. Dieses Programm ist kein *standalone Programm*, welches alleine laufen kann, es braucht eine Umgebung, in der es ausgeführt werden kann. Hosting beschreibt das Bereitstellen einer Umgebung, in der der Service ausgeführt wird. Es gibt vier verschieden Arten einen WCF-Service zu hosten:

- Hosting in einer Anwendung.
- Hosting mit den Internet Information Services (IIS).
- Hosting als Windows Dienst.
- Hosting mit dem Windows Activation Service (WAS).

In Abbildung 2.3 ist der Zusammenhang der einzelnen Komponenten grafisch dargestellt.

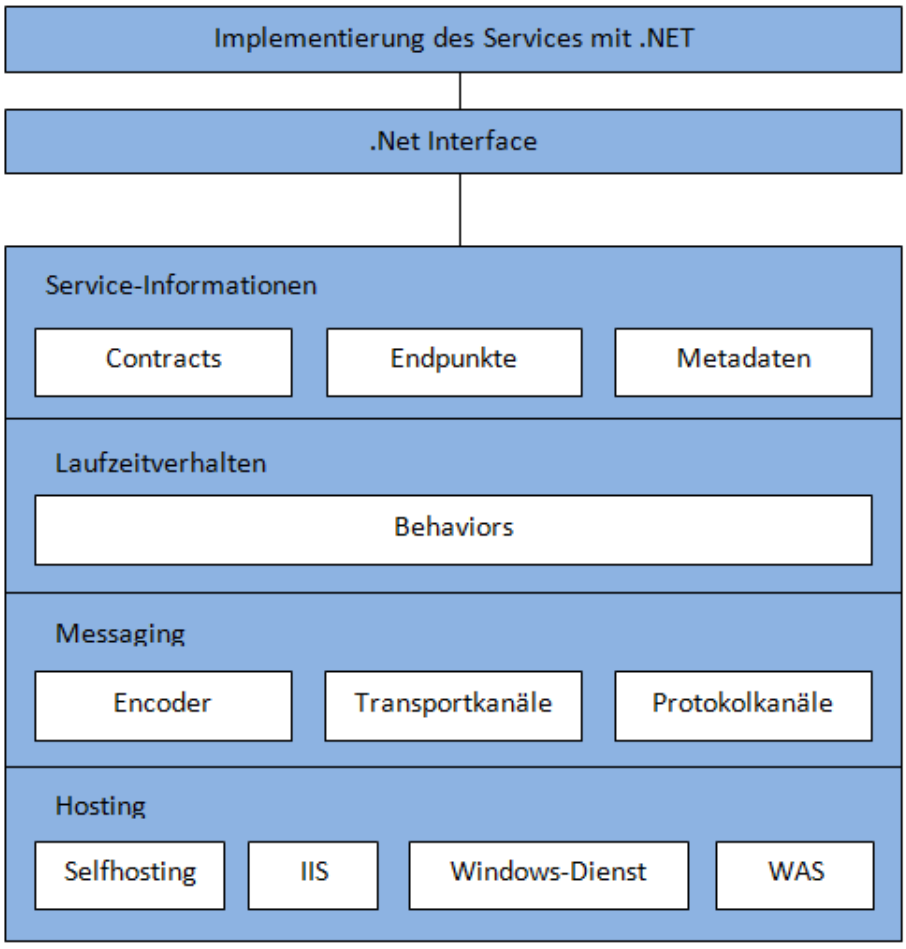

Abbildung 2.3.: Übersicht über die WCF-Komponenten [KH09]

2.3.4. Visual Basic for Applications

Visual Basic for Applications (VBA) ist eine Skriptsprache, die von Visual Basic abgeleitet wurde. Visual Basic stammt aus dem Hause Microsoft. Bis Version 6.0 war die Sprache rein klassenbasiert, ab 7.0 kam das Vererbungsprinzip hinzu, somit ist es ab 7.0 möglich auch objektorientiert zu programmieren. VBA wird von einem Interpreter verarbeitet und ersetzt die früher häufig verwendeten Makrosprachen, was Skriptsprachen in einem Windowsumfeld sind. Alle in VBA geschrieben Programme benötigen eine VBA-Host, in dem sie ausgeführt werden [Sch10].

2.3.5. Component Object Model

Das Component Object Model (COM) ist eine Plattformtechnik aus dem Hause Microsoft um auf verteilte Objekte zugreifen zu können. Das Model wurde 1994 vorgestellt.
Um auf die Objekte zugreifen zu könne werden diese in der Windows Registry registriert und sind somit von allen Anwendungen aufrufbar. Damit die Objekte in der Windows Registry erkannt werden und einzigartig sind bekommen sie sogenannte Global Unique Identifiers (GUIDs). Solche GUIDs sind eindeutige Zahlen, die in 128 Bit kodiert sind [Thu10].
Um nicht alle COM-Objekte, die in der Windows Registry registriert sind, gleichzeitig im Hauptspeicher halten zu müssen hat jedes COM-Objekt einen eigenen Zähler. Anhand dieses Zählers wird bestimmt wie oft das Objekt von anderen Anwendungen referenziert wird. Ist dieser Zähler bei Null, so wird es mit der nächsten Garbage Collection aus dem Hauptspeicher entfernt [Thu10].
Die veröffentlichten Methoden und Objekte werden auf der Binärebene zur Verfügung gestellt, das heißt sie sind nicht abhängig von Programmiersprachen. Sie können also von anderen Programmiersprachen verwendet werden. Um dies zu ermöglichen gibt es gewisse Standards auf die COM-Objekt aufbauen müssen [Thu10].
In COM gibt es drei wesentliche Definitionen [Loo01]:

COM-Schnittstelle: Diese Schnittstellen veröffentlichen die Methoden der Objekte damit andere Anwendungen darauf zugreifen können. Die Standardschnittstelle, welche von Microsoft bereits vordefiniert ist, nennt sich *IUnknown*. Jede weiter Schnittstelle leitet von dieser ab.

COM-Klasse: Jede konkrete Implementierung einer COM-Schnittstelle nennt sich COM-Klasse. Jede Klasse bekommt einen eindeutige GUID in der Windows Registry.

COM-Objekt: Die Instanz einer COM-Klasse wird als COM-Objekt bezeichnet.

Für die Lebensdauer von COM-Objekte gilt folgende Definition:

> *Objekte, die mit New oder CreateObject erzeugt wurden, werden so lange im Arbeitsspeicher gehalten, bis sie durch mindestens eine Objektvariable referenziert werden. Im Umkehrschluss bedeutet dies, dass Objekte automatisch dann zerstört werden - also aus dem Arbeitsspeicher entfernt werden -, wenn sie durch keine Objektvariable mehr referenziert werden.* [LH09, Seite 124]

2.3.6. Microsoft Message Queue

Unter einer Message Queue versteht man eine Warteschlange. Die Implementierung einer Warteschlange aus dem Hause Microsoft nennt man Microsoft Message Queue. Eine Warteschlange zeichnet aus, dass Nachrichten nicht direkt an einen Empfänger, sondern an eine Warteschlange geschickt werden, welche die Nachrichten speichert, bis der Empfänger sie abholt. Dazu steht sie im Gegensatz zu dem *Remote Procedure Call (RPC)*, dort werden Nachrichten direkt an den Empfänger geschickt.
Durch das direkte Senden der Nachrichten an den Empfänger steigt das Risiko eines Fehlers, wenn das aufgerufene System nicht verfügbar ist. Bei einer Warteschlange wird dieses Problem umgangen, da die Nachrichten auf die Warteschlange gelegt werden und nur, wenn das System verfügbar ist werden die Nachrichten auf der Warteschlange gelesen. Das bedeutet die Verfügbarkeit hängt in erster Linie gar nicht von dem System, sondern der Warteschlange ab.
Ein weiterer Vorteil von Warteschlange ist die Skalierbarkeit. Bei hoher Auslastung können mehrere Systeme die Nachrichten von der Warteschlange einlesen und bearbeiten. Bei RPCs ist dies schwieriger zu realisieren, da hier normalerweise direkt Systeme aufgerufen werden.
Die Microsoft Message Queue (MSQM) bietet eine nachrichtenbasierte Kommunikation, die aus einigen Diensten des *Component Object Models (COM)* (siehe Abschnitt 2.3.5) besteht. Es findet eine asynchrone Kommunikation statt, der Client wartet nicht auf die Antwort des Servers und wird so auch nicht blockiert, falls der Server im Moment nicht antwortet. Damit eignet sich Microsoft Message Queue (MSQM) besonders zum Aufbau lose gekoppelter und nachrichtenbasierter Systeme, wie sie in dieser Arbeit verwendet werden. Des weiteren gilt, dass das Nachrichtenformat völlig frei wählbar ist. Einzige Voraussetzung ist, dass die übertragenen Nachrichten serialisierbar sein müssen [KB07].

2.3.7. Java

Java ist eine objektorientierte Programmiersprache von Sun, mit dessen Entwicklung 1990 begonnen wurde. In den Grundstrukturen ähnelt sie sehr stark C++, aber es wurden auch einige Elemente von Smalltalk übernommen, wie zum Beispiel der Bytecode oder die Garbage Collection.
Java wurde hauptsächlich für Internetanwendungen konzipiert, da Javaprogramme im Grunde genommen auf jedem Rechner ausführbar sind. Die Portabilität von Java wird erreicht, in dem man die Javaprogramme zunächst per Compiler in einen Bytecode übersetzt. Dieser Bytecode wird anschließend von einem Interpreter in der jeweiligen Zielplattform ausgeführt. Um die Geschwindigkeit zu steigern werden bei den heutigen Interpretern sogenannte *Just-in-Time-Compiler* eingesetzt, die den Bytecode in einen nativen Maschinencode übersetzen [MSH03].
Java besitzt eine sehr große Entwicklergemeinde (laut Java-Homepage 6,5 Millionen [jav]). Daher gibt es auch viele Open-Source-Produkte. Unter anderem ist der in Abschnitt 2.3.1 beschriebene ESB auf einer Java-Plattform realisiert.

2.3.8. Java Business Integration

Die Java Business Integration gilt auf der Javaplattform als Standard für die SOA-Integration von unterschiedlichen Systemen. Mit Hilfe von Adaptern und Wrappern ist es möglich eine Vielzahl von Technologien in das System einzubinden und sie durch Vergabe von Endpunkten für andere Anwendungen aufrufbar zu machen.

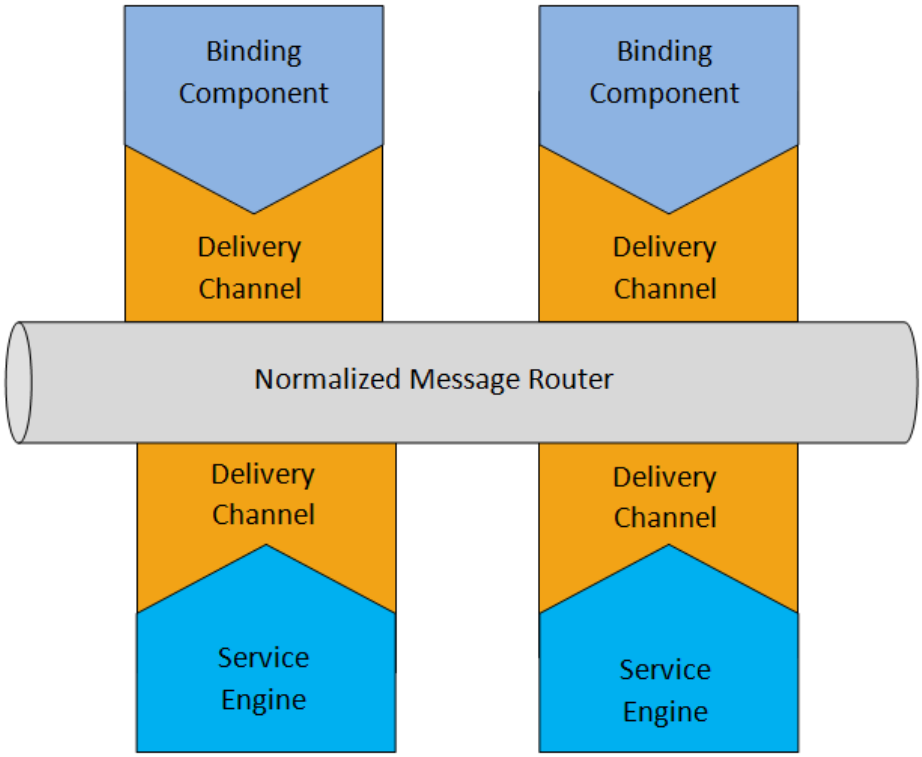

Abbildung 2.4.: Vereinfachte JBI-Komponente [Bibo8]

Die vereinfachte Abbildung 2.4 zeigt die wichtigsten Komponenten des JBI [Mato8] [LSo8] [Bibo8]:

Service Engine: Über eine Service Engine lassen sich beliebige Service Provider an den Normalized Message Router (NMR) anschließen. Diese Service Provider können beispielsweise Nachrichten transformieren, Code ausführen oder weitere Dienste bereitstellen.

Binding Component: Externe SOA-Clients können mit Hilfe der Binding Components an den NMR angeschlossen werden. Die Schnittstellen zur NMR sind mit WSDL beschrieben, die das externe Nachrichtenformat in ein normalisiertes Format transformieren. Das bedeutet eine Service Engine kann über verschiedene Externe Protokolle aufgerufen werden.

Normalized Message Router (NMR): Der Normalized Message Router ist das Bindeglied zwischen Service Engine und den Binding Components. Dabei werden alle Nachrichten der JBI Komponenten über den NMR geleitet. Dadurch sind alle JBI Komponenten lose gekoppelt. Für den Nachrichtenaustausch auf dem NMR werden Normalized Messages verwendet.

Normalized Messages: Die Normalized Messages bestehen aus mindestens zwei Teilen. Zum einem Header, der die Metadaten, wie zum Beispiel Sicherheitsinformationen oder Quality of Services (QoS). Und zum anderen aus den eigentlichen Nutzdaten, dem sogenannten *Payload* welcher den Inhalt der Nachricht enthält. Der Aufbau des *Payloads* entspricht einer XML-Datei, die im XML-Schema (XSD) der WSDL Datei, die zum Nachrichtenaustausch definiert wurde, entspricht. Im dritten Teil der Normalized Message können Anhänge, wie Bilder, enthalten sein.

Delivery Channel: Der Delivery Channel (DC) ist die bidirektionale Verbindung zwischen der Service und dem NMR oder der Binding Component und dem NMR.

2.3.9. SQL Server

Der SQL Server ist ein relationales Datenbanksystem von Microsoft.
SQL steht als Abkürzung für Structured Query Language, was eine Sprache zur Abfrage und Manipulation von Daten einer Datenbank ist.
Der Name SQL Server ist ein Sammelbegriff für verschiedene Elemente, die in ihrer Gesamtheit die Software ausmachen. Im folgenden werden die wichtigsten Elemente kurz zusammengefasst [Kano8]:

Instanz: Der SQL Server wird immer in Instanzen installiert, das heißt auf einem Betriebssystem kann es mehrere Instanzen des SQL Servers geben. Es ist auch möglich verschiedene Versionen des SQL Servers zu installieren.

Datenbankmodul: Das Datenbankmodul ist die wichtigste Komponente des SQL Servers. Hier werden die SQL Anweisungen ausgeführt und die vom Benutzer angelegten Daten gespeichert. Außerdem werden weitere wichtige Aufgaben, wie die Ressourcenverwaltung und das Sicherstellen der Transaktionssicherheit bereitgestellt. Zudem besteht die Möglichkeit zur Backuperstellung.

SQL Server Agent: Der Agent ist ein Dienst, der automatische Aufgaben übernimmt, wie etwa die Wartung der Datenbank.

Systemdatenbanken: Die Systemdatenbanken werden auch als Systemkatalog bezeichnet und beinhalten alle Datenbanken, die das System zur Verwaltung benötigt. Zum Beispiel stellen sie Informationen über den freien Speicherplatz oder die angemeldeten Benutzer zur Verfügung.

In dieser Arbeit wurde der SQL Server 2008 verwendet. Alle in Abschnitt 3.3 aufgeführten Services verwenden ausschließlich die folgenden Datentypen [Kon10]:

NVARCHAR(MAX): Text, der maximalen Länge

DATETIME: Datum und Uhrzeit

INT: Ganze Zahl

SQL Server Management Studio

Das SQL Server Management Studio gilt als das wichtigste Verwaltungswerkzeug für den SQL Server 2008. Es ist abwärtskompatibel, das heißt es können auch SQL Server Instanzen früherer Versionen geöffnet werden [Kon09].
Das Programm bietet eine grafische Oberfläche, doch alle Änderungen, die gemacht werden, werden intern in SQL umgewandelt und auf den jeweiligen Datenbanken ausgeführt.
Mit Hilfe des Management Studios lassen sich alle Datenbanken auf allen SQL Server Instanzen konfigurieren. Außerdem können mit diesem Tool die einzelnen Tabellen bearbeitet werden. Die Funktionalität dieses Werkzeug geht allerdings über das Konfigurieren des Datenbankmoduls hinaus. So können zusätzlich folgende Servertypen administriert werden [Kon10]:

- Analysis Services
- Reporting Services
- Integration Services
- SQL Server Mobile (spezielle Version des SQL Servers für Personal Digital Assistant (PDA) und Pocket Personal Computer (PC))

3. Architektur

In diesem Kapitel wird die neue Architektur des Gesamtsystems vorgestellt. Hauptziel war es den Delmia Process Engineer (DPE) und den Planungstisch (PT) in die eine, auf SOA gestützte, Architektur zu integrieren. Dazu war ein ESB, der die Routingaufgaben für alle verwendeten Systeme übernimmt, gefordert. Außerdem sollten Services erstellt werden, bei denen die Schnittstelle unabhängig von Implementierung des Services ist. Um zu gewährleisten, dass das Gesamtsystem problemlos erweiterbar ist und die Services dynamisch aufrufbar sind soll ein gemeinsames Nachrichtenmodell verwendet werden. Zur Unterstützung der losen Kopplung sollen Message Queues eingesetzt werden.
Der Anfang dieses Kapitels bilden die Workflows, die aus den verschiedenen Services erstellt werden. Diese werden bewusst am Anfang des Kapitels vorgestellt um eine Übersicht über den Ablauf der Layoutplanung zu erhalten.
Um die oben genannte Architektur zu erreichen werden alle dafür notwendigen Komponenten vorgestellt. Angefangen von dem Nachrichtenmodell, das zum Austausch der Daten der verschiedenen Systeme verwendet wird, über die Beschreibung der integrierten Werkzeuge bis hin zur Erklärung der Services, die als Schnittstelle der Werkzeuge dienen. Am Ende des Kapitels befinden sich Informationen zu den Implementierungen aller verwendeten Systeme.

3.1. Workflows

In diesem Abschnitt werden zwei Workflows, der Routing Workflow und der Planungs Workflow, gezeigt und beschrieben. Alle in den Workflows verwendeten Komponenten werden in Abschnitt 3.3 genauer beschrieben.

3.1.1. Routing Workflow

Der Routing Workflow ist in Grafik 3.1 zu sehen. Der Workflow startet, indem er von einer anderen Anwendung aufgerufen wird.
Der Ablauf des Workflows gliedert sich in folgende Schritte:

Ablauf

- Der Workflow wird von einer anderen Anwendung aufgerufen und somit gestartet.

- Danach wird überprüft, ob in der erhaltenen Nachricht bereits Services, die aufgerufen werden müssen, enthalten sind.
- Ist dies der Fall, so wird eine neue Nachricht erstellt, in welcher der erste Service weggelassen wird.
  Sind noch keine Services in der XML-Nachricht enthalten, so wird die Routerlogik des Routing Services aufgerufen. Dieser bestimmt alle Zieladressen der übergebenen Nachricht, fügt diese in eine neue Nachricht ein und schickt diese an den Routing Workflow zurück.
- In beiden Fällen wird der erste Service, der in der Nachricht enthalten ist, aufgerufen.

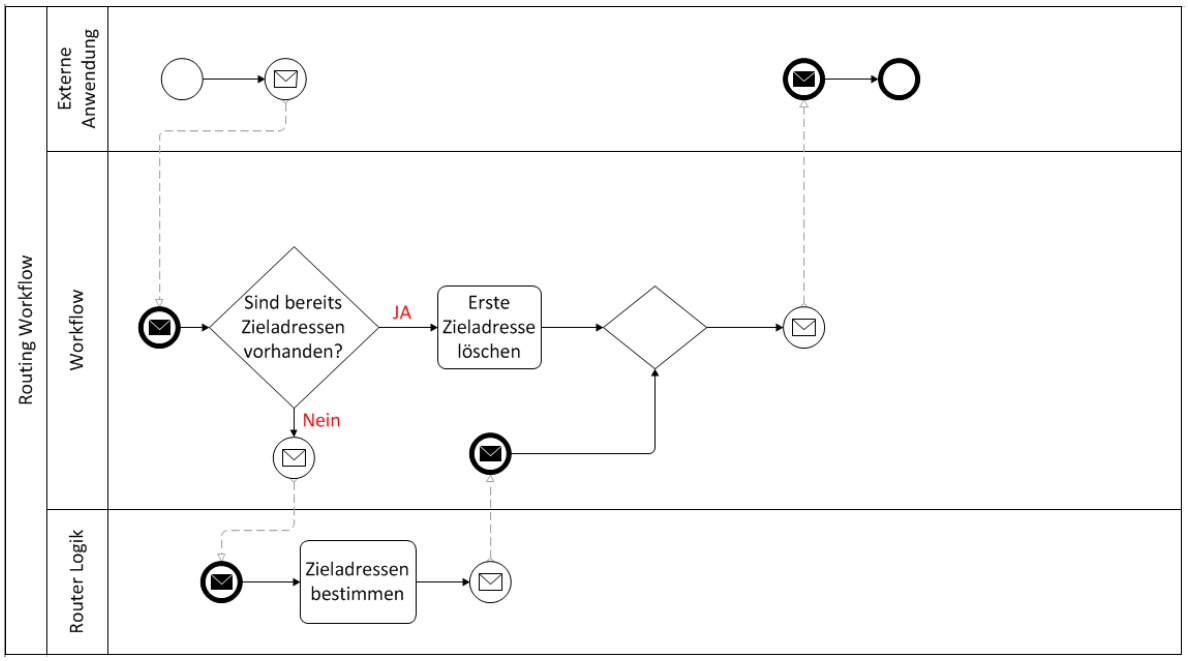

Abbildung 3.1.: Routing Workflow

3.1.2. Planungs Workflow

In dem Planungs Workflow werden alle Schritte der Layoutplanung modelliert. Er wird vom Portal aus gestartet.
Um den Nachrichtenverlauf besser darstellen zu können wurde der Router nicht mit abgebildet. Aus Gründen der Übersichtlichkeit wurden zwei Abbildungen erstellt um den Kompletten Workflow darzustellen, diese sind in Abbildung 3.2 und Abbildung 3.3 zu sehen.

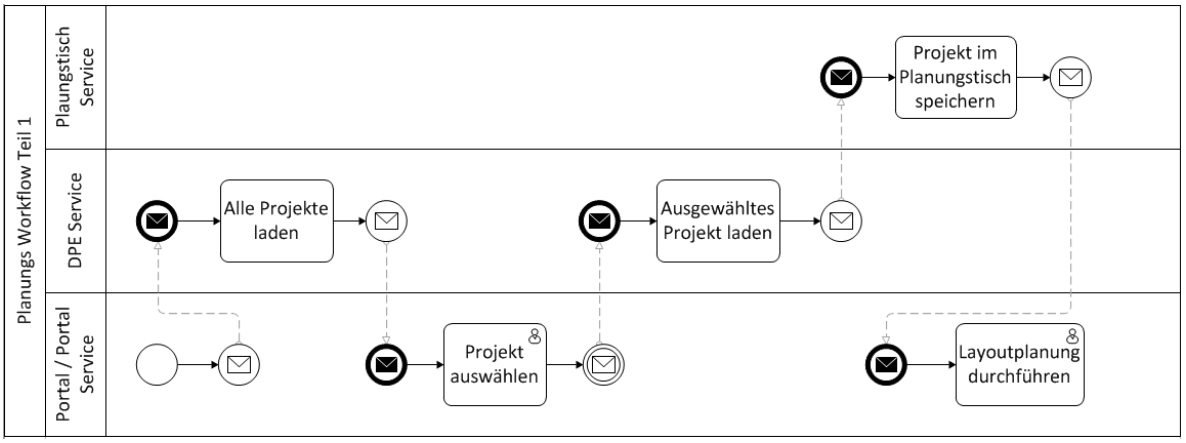

Abbildung 3.2.: Planung Workflow Teil 1

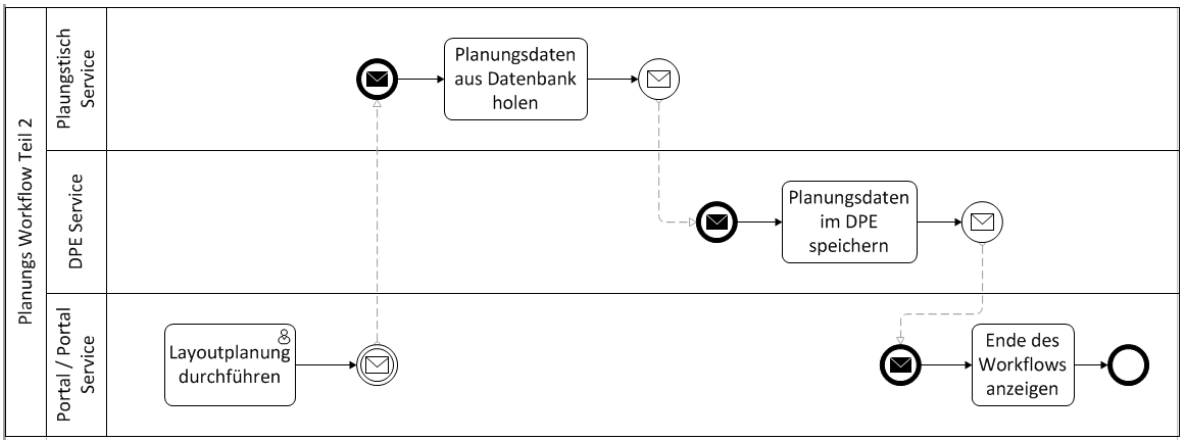

Abbildung 3.3.: Planung Workflow Teil 2

Ablauf

Die einzelnen Schritte des Workflows lauten wie folgt:

- Das Portal ruft den DPE Service auf, der alle Projekte aus dem DPE liest.
- Der DPE Service verpackt die Projekte in eine Nachricht und schickt diese an den Portal Service. Anschließend aktualisiert das Portal die Oberfläche und zeigt den Namen und die ID der Projekte an.
- Der Benutzer wählt eines der anzeigten Projekte aus. Das Portal schickt eine Nachricht an den DPE Service.
- Der DPE Service liest das Projekt aus dem DPE aus und schickt es in einer Nachricht an den Planungstisch Service.

- Der Planungstisch Service lädt das Projekt mit Hilfe des Access Services in die Datenbank des Planungstisches.
- Der Planungstisch Service schickt eine Nachricht an den Portal Service, dieser gibt die Nachricht über die Datenbank an das Portal weiter, welches anschließend die Oberfläche aktualisiert.
- Der Benutzer macht die Layoutplanung am Planungstisch. Wenn er diese beendet hat wird eine Nachricht vom Portal an den Planungstisch Service geschickt. Dieser liest das Ergebnis der Layoutplanung mit Hilfe des Access Services aus der Accessdatenbank des Planungstisches aus und schickt es mit einer Nachricht an den DPE Service.
- Der DPE Service speichert das Ergebnis im DPE und schickt eine Nachricht an den Portal Service.
- Über das Portal wird dem Benutzer das Ende der Layoutplanung mitgeteilt.

3.2. XML Schema

Das XML Schema, das im Listing A.1 zu sehen ist, stellt den Aufbau der Nachrichten dar. Eine Beispielnachricht findet sich im Listing A.2. Alle versendeten Nachrichten innerhalb des ESB genügen diesem Schema. Es unterteilt sich in zwei verschiedene Bereiche. Im ersten Bereich finden sich Metadaten wieder. Diese dienen als Nachrichtenkopf und stellen Informationen wie Ziel der Nachricht oder den Nachrichtentyp bereit. Im zweiten Bereich stehen die tatsächlichen Nutzdaten, die für die Planung in den verschiedenen Systemen erforderlich sind.

3.2.1. Metadaten

Um das Routing zu ermöglichen und eine spätere Nachrichtenverfolgung bereitzustellen wurden folgende Typen eingeführt:

1. **MessageType:** Der MessageType gibt Aufschluss über die aktuelle Planungsphase. Je nach MessageType wird im Router entschieden wohin die entsprechenden Nachrichten geschickt werden müssen. Der MessageType lässt sich wie folgt aufschlüsseln:

   0: Alle Projekte von dem DPE zum Portal

   1: Das ausgewählte Projekt vom DPE in PT laden

   2: Das aktuelle Projekt vom PT im DPE zurückspeichern

2. **RoutingInfos:** Diese Informationen dienen ausschließlich der Kommunikation. Jedes System schickt seine Nachricht an den Router. Dieser entscheidet anhand der Routingstrategie wohin die Nachricht weitergeleitet werden muss. Somit müssen die Systeme lediglich die Adresse des Routers kennen und nicht die komplette Systemarchitektur mit den jeweiligen Endpunkten. Wird ein System ausgetauscht, so muss nur die neue Adresse in der Routingtabelle des Routers aktualisiert werden.

3. **Message-ID und Messageflow-ID:** Jede Nachricht ist im Gesamtsystem einzigartig. Um das zu erreichen wird in jeder Nachricht eine Message-ID vergeben. Diese setzt sich aus dem Kürzel des jeweiligen Quellsystems und dem Zeitstempel, an dem die Nachricht verschickt wird, zusammen. Die Message-ID wird im Messageregistration Service erzeugt. Siehe Abschnitt 3.3.9.
   Um die Zusammengehörigkeit verschiedener Nachrichten zu definieren kann eine Messageflow-ID vergeben werden. Diese Messageflow-ID entspricht dann der Message-ID der ersten Nachricht und wird im Messageflowregistration Service erzeugt. Siehe Abschnitt 3.3.10.

4. **RoutingDestination:** Hier werden die ID und die URI der Ziele der Nachricht gespeichert. Der Routing Service füllt die Felder und aktualisiert sie bei Bedarf.

3.2.2. Nutzdaten

Die Wurzel der Nutzdaten ist das Projekt. Projekte werden in den PT geladen und dort eine Layoutplanung durchgeführt. Da das XML Schema generisch erstellt wurde, um auf Erweiterbarkeit (siehe Abschnitt 5.2.1) zu achten, wurde noch eine zusätzliche Stufe eingeführt, die *Resourcen*.
In Abschnitt 3.3.2 wird der DPE vorgestellt. Wie dort erklärt wird beruht die Struktur des DPE auf den drei Kernbereichen:

- Produkte
- Prozesse
- Ressourcen

Das momentane Nachrichtenmodell berücksichtigt nur die Ressourcen, da der Planungstisch (siehe Abschnitt 3.3.3) nur Daten aus diesem Bereich benötigt. Doch da sichergestellt werden sollte, dass das Nachrichtenmodell um die anderen zwei Bereiche erweitert werden kann, ohne die vorhandenen Implementierungen der momentan vorhandenen Systeme ändern zu müssen, wurde zusätzliche die Stufe *Resourcen* hinzugefügt. Durch dieses Vorgehen werden die zwei, noch nicht vorhandenen Stufen, behandelt als wären sie leer. Wenn sie später gefüllt sind werden sie genauso ignoriert wie jetzt.

Folgende Daten stehen unter dem Knoten *Resourcen*, da diese vom Planungstisch verwendet werden:

1. **Position:** Hier werden die Positionen auf der X-, Y- und Z-Achse für die einzelnen Planungsobjekte gespeichert.
2. **Module:** Das sind die einzelnen Planungsobjekte. Hier wird der Name und die ID des Objekt, sowie die einzelnen Positionsdaten gespeichert.
3. **ModuleType:** Die Moduletypen entsprechen Klassen. Jeder Moduletyp kann mehrere Module unter sich haben. Ein Module ist sozusagen eine Instanz eines Moduletyps. Hier werden verschiedene Informationen gespeichert, die später am PT notwendig sind, um die Moduletypen richtig darzustellen. Beispiele für die zusätzlichen Informationen sind:
   - Ein Bild, das im Planungstisch angezeigt wird, wenn ein Module in das Layout gezogen wird.
   - Das Computer-Aided Design (CAD) File, das hinter dem Modul liegt.

3.3. Systemübersicht

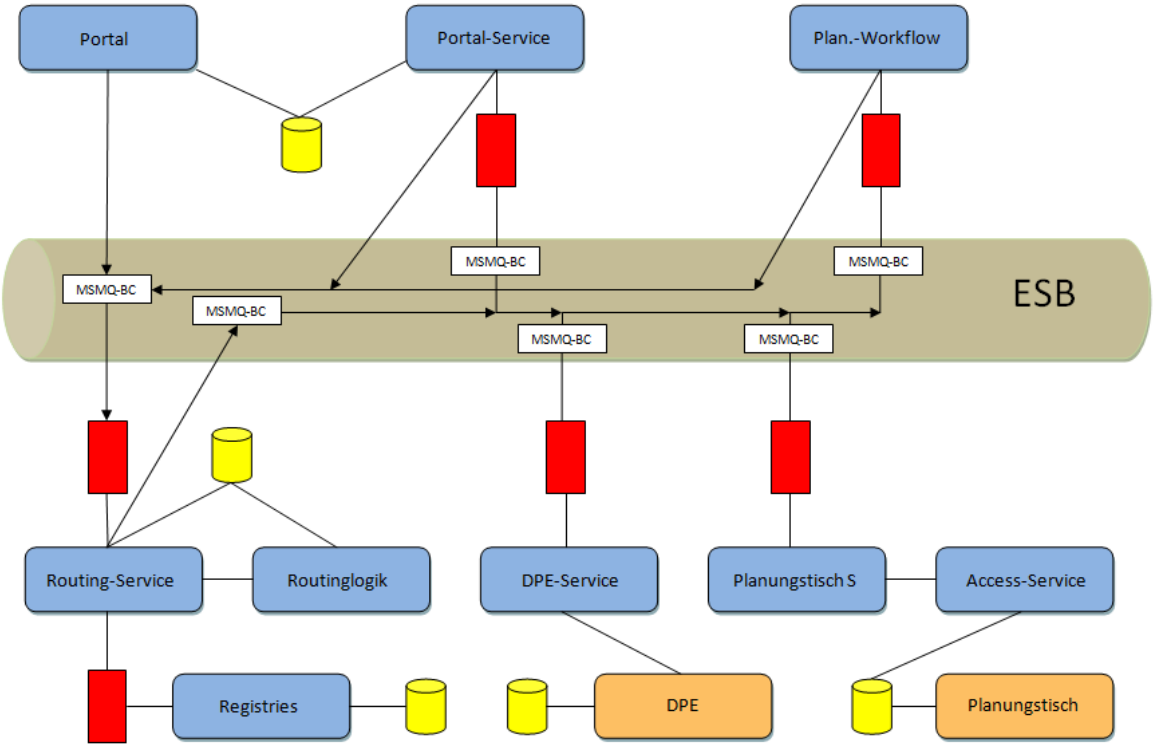

Abbildung 3.4.: Systemübersicht

Abbildung 3.4 zeigt die Systemübersicht mit allen verwendeten Systemen. Rot eingefärbt sind die lokalen Warteschlangen der einzelnen Services. Die gelb gekennzeichneten Zylinder stellen die Datenbanken dar, auf die zugegriffen wird. Die Systeme, die integriert werden sollen, werden in orange gehalten. Blau markiert sind alle Services, das Portal und der Workflow.
Eine genauere Beschreibung der verwendeten Systeme folgt in den folgenden Abschnitten.

3.3.1. Portal und Portal Service

Das Portal ist eine Webseite, auf der es möglich ist Workflows zu starten und dessen Fortschritte nachzuvollziehen. Im Moment ist im Portal nur ein Workflow integriert, der Planungs Workflow. Beim Aufbau der Webseiten wurde darauf geachtet, dass auch weitere Workflows in diesem Portal gestartet werden können (siehe Abschnitt 5.2.2).
Der Portal Service ist, wie die anderen Services auch, ein WCF Service. Er wird als Windows Dienst gehostet. Er besitzt eine eigene Warteschlange, die er ständig abhört. Seine Schnittstelle erwartet Nachrichten, die dem gemeinsamen XML Schema genügen.

Datenbankaufbau

Die Abbildung 3.5 zeigt das gemeinsam genutzte Datenbankschema des Portals und des Portal Services. Die Datenbank besteht aus mehreren Tabellen:

Updates: Die Tabelle *Updates* wird sowohl von dem Portal, als auch von dem Portal Service benutzt. Bekommt der Portal Service eine neue Nachricht vom Router, dann löscht der Portal Service den Inhalt der Tabelle und schreibt die aktuelle Nachricht in diese Tabelle.
Im Portal ist ein Timer integriert, der alle fünf Sekunden ausgelöst wird und die Tabelle *Updates* auf Änderungen überprüft. Wurde eine neue Nachricht in die Tabelle eingefügt, dann liest das Portal diese aus und passt die Oberfläche entsprechend an.

Currentmessage: In dieser Tabelle werden alle Nachrichten gespeichert, die vom Portal aus gesendet werden. Hier wird der Status des ausgeführten Workflows hinterlegt.

Texts: In dieser Tabelle befinden sich alle Statusmeldungen, die auf dem Portal angezeigt werden. So können diese vom Benutzer ohne großen Aufwand in der Datenbank geändert werden.

Modules: Die Tabelle *Modules* beinhaltet alle Workflows, die gestartet werden können. Durch das Erstellen der Tabelle wird es vereinfacht weitere Workflows, ohne großen Programmieraufwand, auszuführen.

Users: In dieser Tabelle werden alle verfügbaren Benutzer des Portals mit Passwort abgespeichert. Dadurch wird verhindert, dass feste Benutzer im Programm abgespeichert werden und die Benutzerverwaltung nur mit Hilfe von Programmieraufwand möglich ist.

Activeusers: Die Tabelle speichert alle Benutzer, die momentan am Portal angemeldet sind. Zusätzlich wird noch abgespeichert welchen Workflow sie zur Zeit ausführen.

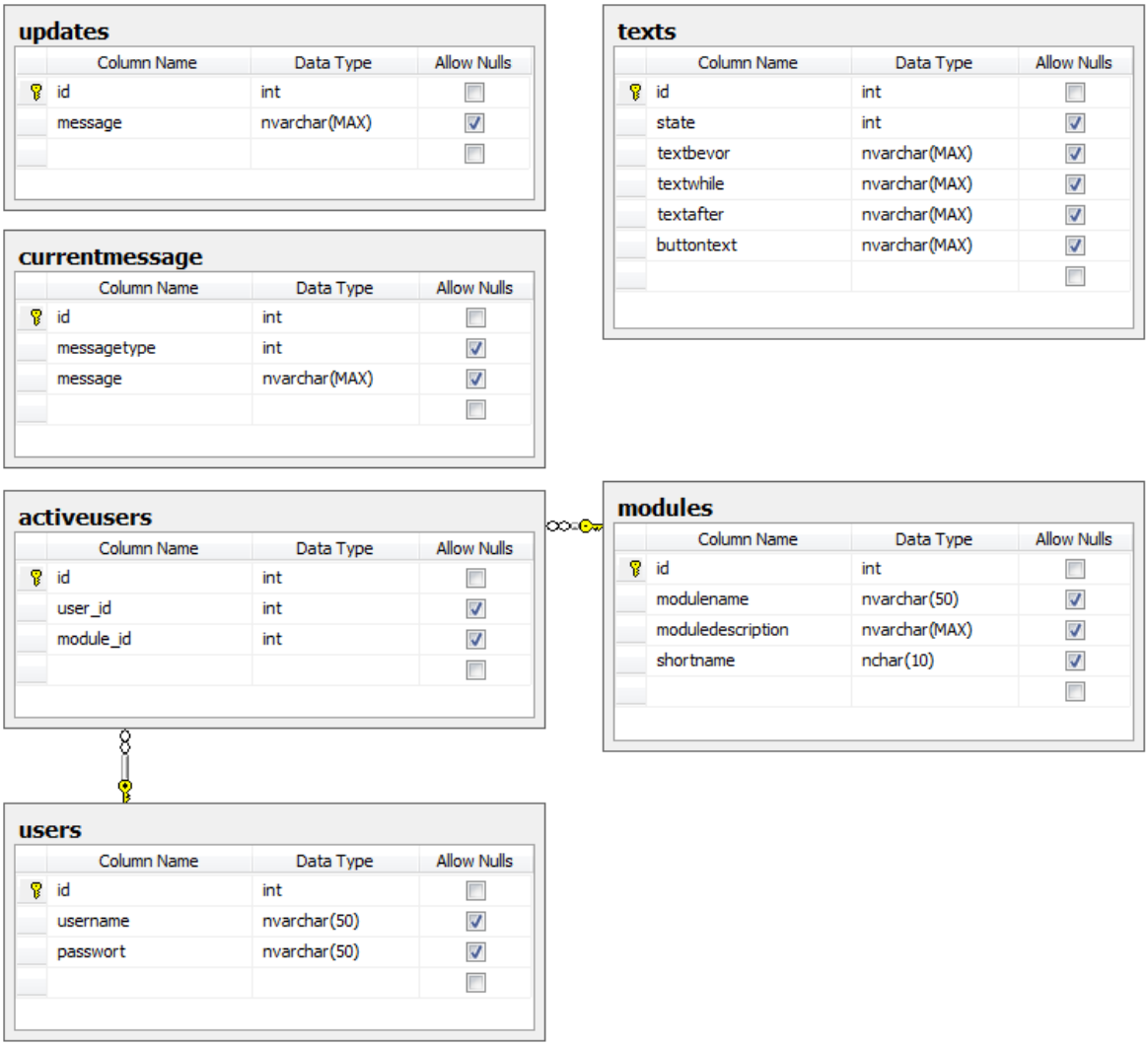

Abbildung 3.5.: Datenbankschema Portal und Portal Service

Ablauf Portal

Im Folgenden wird der Ablauf anhand von Bildern beschrieben.

Anmelden Das Anmelden geschieht in einer klassischen Anmeldeoberfläche. Auf dieser wird der Benutzername und das Passwort mit den Einträgen der Tabelle *Users* aus der Datenbank verglichen. Stimmen diese überein, so wird der Benutzer auf die nächste Seite weitergeleitet. Auf dieser entscheidet sich der Benutzer welchen Workflow er starten oder fortführen möchte.

Workflow auswählen In Abbildung 3.6 ist die Oberfläche, die alle Workflows auflistet, zu sehen. Auf dieser Seite entscheidet sich der Benutzer für einen Workflow. Wenn er diesen selektiert, wird überprüft, ob eine Instanz dieses Workflows bereits ausgeführt wird oder nicht. Falls ja, hat der Benutzer die Möglichkeit, den aktuellen fortzuführen oder einen neuen zu starten (siehe Abbildung 3.7).
Sind alle Instanzen des Workflows beendet, dann wird der Benutzer automatisch zur Startseite des Workflows weitergeleitet.

Abbildung 3.6.: Workflowübersicht

Workflow starten Als erstes wird eine Nachricht an den Router geschickt. Die Nachricht ist vom MessageType 0, d.h. der Router erkennt, dass die Nachricht zuerst an den DPE Service geschickt werden muss, der dort alle vorhandenen Projekte ausliest, diese in eine Nachricht packt, welche dann an den Router geschickt wird, der diese an den Portal Service weiterleitet.
Der Portal Service schreibt die Nachricht in die Datenbank, aus der sie das Portal wieder ausliest und die Oberfläche entsprechend aktualisiert.

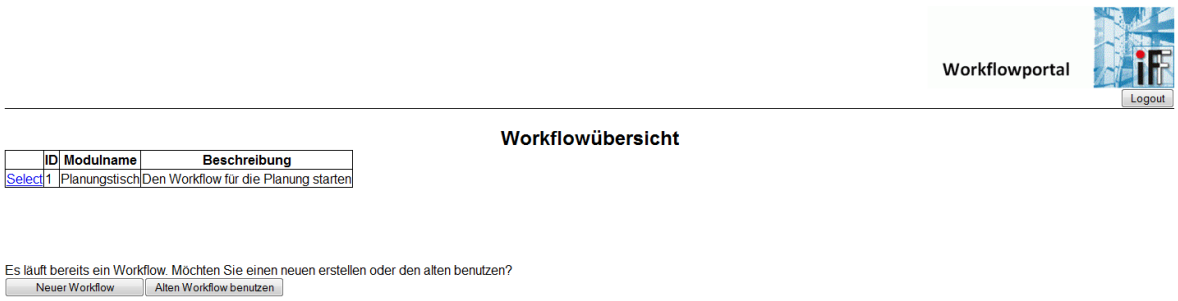

Abbildung 3.7.: Aktiver Workflow

Planung starten Abbildung 3.8 zeigt die Oberfläche nach dem Laden aller Projekte aus dem DPE. Der Benutzer wählt ein Projekt, das in den Planungstisch geladen werden soll.

Um das Projekt in den Planungstisch zu laden wird eine Nachricht vom MessageType 1 an den Router geschickt. Dieser entnimmt aus seiner Routingtabelle, dass die Nachricht zuerst an den DPE Service, dann an den Planungstisch Service und zuletzt an den Portal Service geschickt werden muss.
Im DPE Service wird das ausgewählte Projekt geladen, anschließend wird an den Planungstisch Service weitergeleitet, der den Access Service aufruft mit dessen Hilfe das Projekt in die Accessdatenbank des Planungstisches geschrieben wird. Um dem Benutzer das Ende des Ladevorgangs mitzuteilen wird vom Planungstisch Service eine Nachricht, über den Router, an den Portal Service geschickt, der die neue Nachricht in die Datenbank schreibt. Von dort wird sie vom Portal ausgelesen und die Oberfläche aktualisiert.
Nun kann der Benutzer am Planungstisch die Layoutplanung durchführen.

Abbildung 3.8.: Planung starten

Planung beenden Der Benutzer hat die Layoutplanung beendet und kehrt zum Portal zurück um das Resultat der Planung im DPE zu speichern. Die dafür vorgesehene Oberfläche ist in Abbildung 3.9 zu sehen.
Es wird eine Nachricht vom MessageType 2 an den Routing Service gesendet. Anhand seiner Routingtabelle erkennt der Service, dass die Nachricht zuerst an den Planungstisch Service, danach an den DPE Service und zum Abschluss an den Portal Service, weiterleiten muss.
Der Planungstisch Service ruft den Access Service auf, mit dessen Hilfe das Ergebnis der Layoutplanung aus der Accessdatenbank des Planungstisches gelesen wird und dem Planungstisch Service zurückgegeben wird. Der Planungstisch Service packt das Ergebnis in eine Nachricht und schickt diese über den Router an den DPE Service, der das Ergebnis der Layoutplanung im DPE abspeichert und mit Hilfe des Routers den Portal Service aufruft. Der Portal Service schreibt die angekommene Nachricht in die Datenbank, aus der sie vom Portal ausgelesen und die Oberfläche aktualisiert wird.

Ablauf Portal Service

Der Ablauf des Portal Service gliedert sich in folgende Schritte:

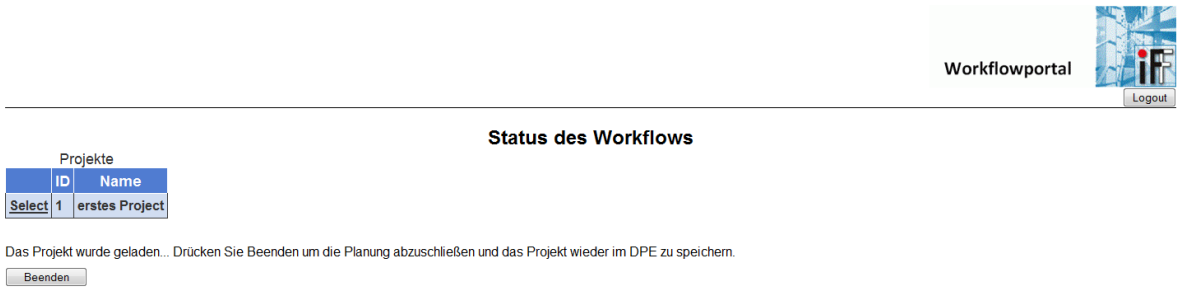

Abbildung 3.9.: Planung beenden

- Der Router schickt eine Nachricht an die Warteschlange des Portal Service.
- Da der Portal Service die Warteschlange ständig abhört, erkennt er die ankommende Nachricht und nimmt sie entgegen.
- Der Service löscht den kompletten Inhalt aus der Tabelle *Updates*.
- Nun schreibt er die empfangene Nachricht in die Tabelle *Updates*. Da die Tabelle vorher gelöscht wurde ist sichergestellt, dass sich darin genau ein Eintrag, die aktuelle Nachricht, befindet.
- Das Portal prüft regelmäßig den Inhalt der Tabelle *Updates* und erkennt daher, dass ein neuer Eintrag in die Tabelle eingefügt wurde.
- Das Portal liest den neuen Eintrag aus der Tabelle und passt die Oberfläche entsprechend der aktuellen Nachricht an.

3.3.2. Delmia Process Engineer

Der Delmia Process Engineer (DPE) ist ein Werkzeug, das in der Planung verwendet wird. Der DPE wurde von dem Unternehmen Dessault Systems entwickelt.
Er wird in folgende Module aufgeteilt (Delmia Produktbeschreibung):

Product Process Resource (PPR) Navigator: Dieses Modul dient zur Projektinitialisierung unter den Planungsprämissen und stellt den Planungszusammenhang zwischen Produkt, Prozess und Ressource dar. Der PPR Navigator dient als zentraler Punkt für die unternehmesweite Prozessplanungsdatenbank.

Product Evaluation: In diesem Modul stehen Funktionen zur Produktevaluierung zur Verfügung.

Process- & Resource Planning: Das Modul wird zur Planung von Prozessen und Ressourcen verwendet.

Layout Planning: In diesem Modul werden 3D-Layoutplanungen erstellt und die Ergonomie von Arbeitsplätzen untersucht.

Standard Time Measurement: Mit Hilfe dieses Moduls werden Fragen bezüglich der Zeitwirtschaft gelöst.

Work Load Balancing: In diesem Modul wird die Auslastung der Arbeitsplätze untersucht.

Automotive Line Balancing: Das Modul stellt Methoden zur Analyse der maschinellen Auslastung bereit.

In allen diesen Modulen werden folgende Grundstrukturen verwendet:

- Produkte
- Prozesse
- Resourcen

In jedem dieser Kernbereiche können verschiedene Plantypen definiert werden. Diese und dessen Verlinkungen bilden die unternehmensspezifische Planungsmethode. Zur übersichtlichen Verwaltung erhält jeder Plantyp, bzw. dessen Verlinkungen, eine Versionsnummer und jede Version wiederum kann mit einem Status versehen werden (Delmia Produktbeschreibung).

Mit Hilfe der Plantypen wird ein möglichst genaues Abbild der Unternehmesstruktur geschaffen. Dadurch wird der komplette Produktentwicklungszyklus, angefangen vom konzeptionellen Design bis hin zur Verbesserung, unterstützt. Dabei können folgende Planungszenarien abgebildet werden (Delmia Schulungsunterlagen):

- Neues Produkt/Neues Layout
- Neues Produkt/Vorhandenes Layout
- Neues Produkt/Bestehende Fertigung
- Vorhandenes Produkt/Neues Werk
- Vorhandenes Produkt/Vorhandenes Layout
- Vorhandenes Produkt/Bestehende Fertigung

Durch das Zusammenspiel von Produkt, Prozess und Resourcen sinkt das Risiko von Fehlern während der Planung. Außerdem wird eine detaillierte Übersicht der anfallenden Kosten, benötigte Produktionsfläche und der benötigten Anzahl von Mitarbeitern, bereitgestellt. Diese Informationen stehen schon in der frühen Phase der Planung zur Verfügung.
Durch das Einbeziehen der Kernbereiche Produkt, Prozess und Ressource werden weitere Vorteile in folgenden Bereichen generiert:

- Optimierung des Materialflusses
- Modularisierung des Produktkonzepts

- Erhöhte Planungsgeschwindigkeit
- Erhöhung der Produktionsflexibilität
- Verbesserte Produktqualität
- Paralleles Arbeiten wird ermöglicht

In dem PPR-Modul werden Produkte, Prozesse und Resourcen in einer Baumstruktur angezeigt (siehe Abbildung 3.10). Die Baumstruktur wird durch das Verschachteln der Plantypen erzeugt.
Zusätzlich bietet der DPE die Möglichkeit VBA-Skripte anzulegen. Diese werden nach dem Start im eigenen Skriptinghost ausgeführt. Außerdem ist es möglich jedem Plantyp ein eigenes Kontextmenü zuzuordnen, das individuell erstellt werden kann. Dadurch können jedem Plantyp spezielle Funktionen, wie das Starten selbst erstellter VBA-Skripte, zugeordnet werden.

3.3.3. Planungstisch

Der Planungstisch (siehe Abbildung 3.11) ist ein Werkzeug zur Layoutplanung, das 3D-Modelle verwendet. Der Planungstisch wurde speziell für das Entwickeln in Gruppen konzipiert. Aufgrund der 3D-Visualisierung können Änderungen sofort im Team diskutiert und evaluiert werden [WZ09] [BS05].

Mit kleinen reflektierenden Klötzchen, den so genannten Bricks, können einzelne Objekte auf dem Planungstisch verschoben werden. Hierzu wird ein Brick auf ein Objekt gestellt. Das System erkennt den Brick, der somit mit dem Objekt verknüpft wird und eine Einheit bildet. Durch verschieben des Bricks wird nun auch das Objekt verschoben. Um die Verbindung von Objekt und Brick wieder zu lösen, muss der Brick kurzzeitig abgedeckt werden. Neue Objekte werden hinzugefügt, in dem man den Brick auf die Objektklassen stellt, die sich am Rand des Layouts befinden und anschließend das Objekt innerhalb des Layouts platziert [Auto3].
Durch die genauen Abmessungen der Fabrikhallen und den exakt hinterlegten Größen der verschiedenen Objekte lässt sich sehr genau beurteilen, ob ein Objekt an die gewünschte Position passt. Dabei können verschiedene Stufen betrachtet werden, wie zum Beispiel der Boden, die Tisch- oder die Regalhöhe. Diese Funktion wird vor allem bei Objekten, dessen Unterbau kleiner ist als der Überbau, eingesetzt.
Als Kernstück des Planungstisches wird *I-Plant* benutzt. Diese Software dient als Oberfläche des Planungstisches und ist ausschließlich über die angeschlossene relationale Datenbank konfigurierbar. Im Planungstisch wird eine Accessdatenbank verwendet, in der nicht nur Layoutdaten, sondern auch noch zusätzliche Informationen zu den Objekten, gespeichert werden. Um die Informationen komfortabel eingeben zu können, besteht die Möglichkeit eigene Formulare in der Datenbank zu hinterlegen.

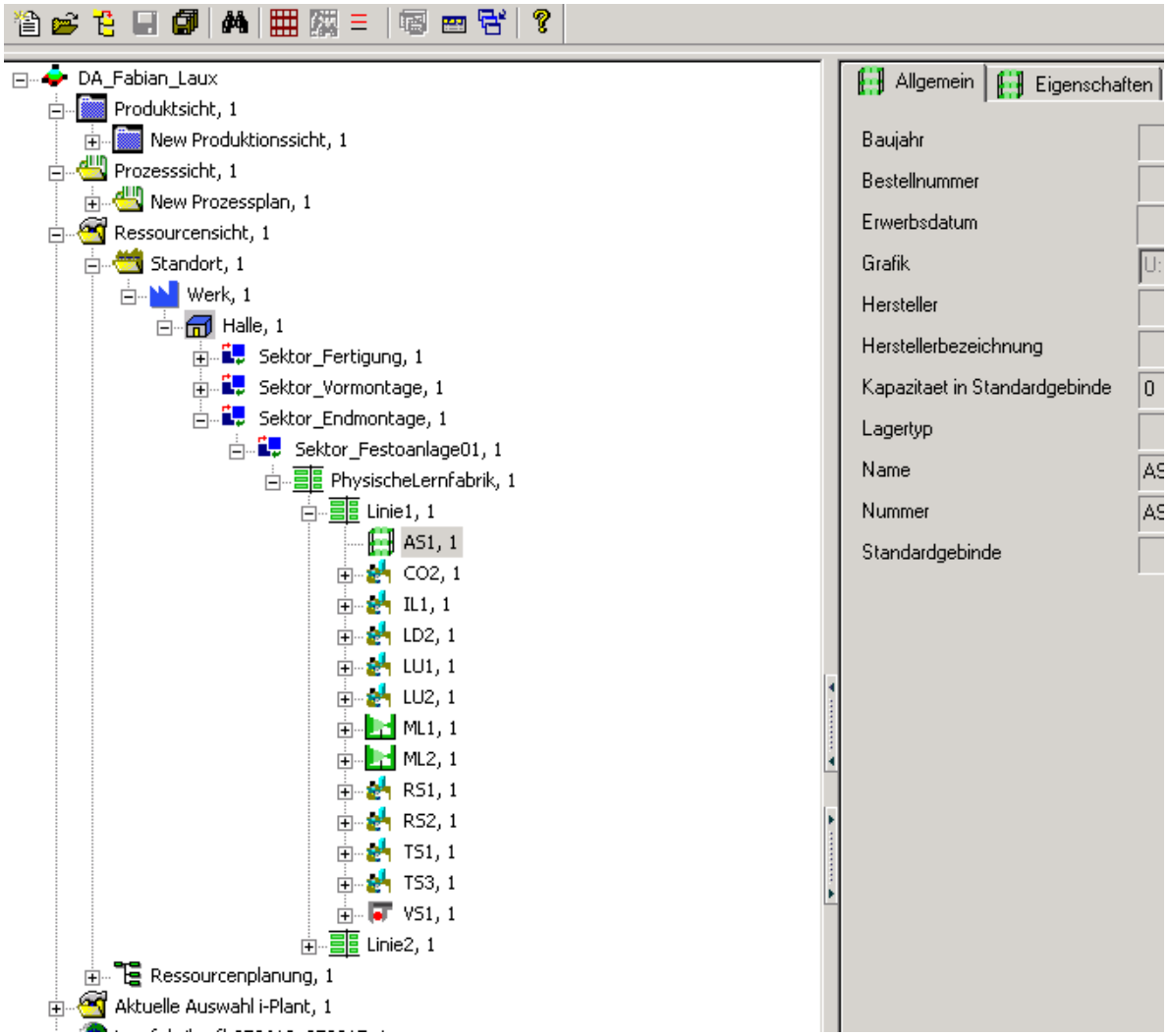

Abbildung 3.10.: Delmia Baumstruktur

3.3.4. Open ESB

Der Open ESB ist der zentrale Punkt der eingesetzten Architektur. Er dient als Dienstplattform für alle systemrelevanten Dienste. Nicht nur die Routerlogik läuft auf ihm, sondern auch die Workflows, wenn diese aufgerufen werden.
Die Workflows werden in Business Process Execution Language (BPEL) ausführbar gemacht. Um die BPEL Prozesse von anderen Anwendungen aufrufbar zu machen, wird JBI verwendet und somit Endpunkte generiert. In dieser Arbeit dienen Message Queues als Endpunkte. Da es sich um Microsoft Message Queues handelt wird das MSMQ-BC verwendet um die Prozesse an diese Endpunkte zu binden.
Alle Nachrichten, die an die Routerlogik geschickt werden kommen bei ihm an.

Abbildung 3.11.: Übersicht über den Planungstisch

3.3.5. DPE Service

Der DPE Service dient als Schnittstelle zum DPE für ankommende Nachrichten vom ESB. Bedauerlicherweise war der DPE während dieser Arbeit aufgrund von Softwareproblemen nicht verfügbar, weshalb ersatzweise eine SQL Server Datenbank mit einem ähnlichen Datenbankschema für das Auslesen und Speichern der Daten verwendet wurde. In dieser Datenbank werden die Daten der Layoutplanung gespeichert in die Datenbank des Planungstisches geladen, dort bearbeitet und wieder in der Datenbank des DPE gespeichert.
In den nächsten Abschnitten wird der geplante Ablauf mit DPE und der momentane Ablauf mit der SQL-Datenbank geschildert.

Geplanter Ablauf mit DPE

Der DPE hat keine Schnittstelle, auf die man von anderen Anwendungen zugreifen kann, deswegen muss der Planungs Workflow vom DPE gestartet werden.

Aus dem DPE heraus sollte ein VBA-Skript gestartet werden, das über eine COM-Schnittstelle den Router aufruft. Um Nachrichten an den DPE zu schicken, wurde überlegt, das Skript in einer Endlosschleife laufen zu lassen, die ständig überprüft, ob sich an gewissen Pfaden neue XML-Files befinden. Falls ja, werden diese eingelesen und anhand des Aussehens der Files erkennt das Skript welche Aktionen es starten muss.

Alle Projekte vom DPE in das Portal laden Das Laden und Verschicken der Projekte ist in folgende Schritte gegliedert:

- Der Benutzer führt im DPE das VBA-Skript zum Laden aller Projekte aus.
- Das Skript liest den Namen und die ID aller Projekte, die im DPE gespeichert sind.
- Anschließend werden die Daten in ein XML-File geschrieben.
- Das XML-File wird in einen String umgewandelt.
- Es wird eine Instanz der Klasse *VBAInterface* erzeugt, die ein in C# erstelltes COM-Objekt ist.
- Der Klasse wird der umgewandelte XML-String übergeben.
- Die Klasse wandelt den String wieder in ein XML-File um und ruft damit den Router auf, der seinerseits erkennt, dass die Nachricht an das Portal weiter geleitet werden muss.

Ausgewähltes Projekt in den Planungstisch laden Der Benutzer wählt ein verfügbares Projekt im Portal aus, worauf der DPE Service folgende Schritte ausführt:

- Der DPE Service erhält eine Nachricht. Diese Nachricht wurde von dem Portal verschickt und über den Router an den DPE Service weitergeleitet.
- Anhand der Nachricht erkennt der DPE Service, dass das übergebene Projekt komplett geladen werden soll.
- Der Service schreibt die XML Nachricht in den lokalen Pfad *C:\XML\load.xml*.
- Da das VBA-Skript des DPE in einer Endlosschleife läuft und diesen Pfad ständig überprüft, erkennt es das neue File und liest es ein.
- Nach dem Einlesen wird die Datei wieder gelöscht.
- Anhand der ID aus der Datei wird das richtige Projekt geladen.
- Anschließend wird das Projekt in eine neue Nachricht verpackt.
- Diese Nachricht wird mit Hilfe der *VBAInterface*-Klasse an den Router geschickt.
- Dieser leitet die Nachricht an den Planungstisch Service und den Portal Service weiter.

Speichern der Daten nach Planungsende Hat der Benutzer die Layoutplanung beendet hat, dann betätigt er den Speichern-Knopf im Portal. Der Ablauf beim Speichern sieht dann wie folgt aus:

- Der DPE Service erhält eine Nachricht mit den aktuellen Daten. Diese Nachricht wurde von dem Planungstisch Service verschickt und mit Hilfe des Routers an den DPE Service weitergeleitet.
- Der DPE Service erkennt anhand der Nachricht, dass die Daten im DPE gespeichert werden müssen und schreibt sie daher in den lokalen Pfad *C:\XML\save.xml*.
- Das VBA-Skript des DPE erkennt die neue Datei, liest sie ein und löscht sie anschließend.
- Die Daten aus der XML-Datei werden von dem VBA-Skript im DPE gespeichert.
- Nach dem Speichern erzeugt das Skript eine neue Nachricht und schickt diese mit Hilfe der *VBAInterface*-Klasse an den Router.
- Dieser leitet die Nachricht an den Portal Service weiter. Die Nachricht wird in der Datenbank des Portal Services gespeichert und anschließend vom Portal ausgelesen.
- In der Oberfläche des Portals wird angezeigt, dass die Layoutplanung nun abgeschlossen ist.

Momentaner Ablauf ohne DPE

In Abbildung 3.12 ist das Datenbankschema, das verwendet wird um die Struktur im DPE nachzubilden, zu sehen. Der Zwischenschritt über die COM-Klasse *VBAInterface* kann weggelassen werden, da der DPE Service auf die SQL Server Datenbank direkt zugreifen kann. Dennoch sind die Schritte und Abläufe vergleichbar.

Alle Projekte von der Datenbank in das Portal laden Nach dem Starten des Workflows über das Portal werden beim DPE Service folgende Schritte ausgeführt:

- Der DPE Service erhält eine Nachricht vom Router, der die ursprüngliche Nachricht vom Portal weitergeleitet hat.
- Anhand der Nachricht wird erkannt, dass alle Projekte aus der Datenbank geladen werden sollen.
- Der Service stellt eine Verbindung zur Datenbank her und liest den Namen und die ID aller Projekte aus.
- Die ausgelesenen Projekte werden in eine XML-Nachricht verpackt.
- Diese Nachricht wird an den Router geschickt.

- Dieser erkennt aufgrund seiner Routingtabelle, dass er die Nachricht an den Portal Service weiterleiten muss. Dort schreibt der Portal Service die Nachricht in seine Datenbank, auf die das Portal ebenfalls zugreift und die Nachricht wieder ausliest.
- Das Portal zeigt die Namen und IDs aller Projekte aus der eingelesenen Nachricht an.

Ausgewähltes Projekt in den Planungstisch laden Der Benutzer wählt ein verfügbares Projekt im Portal aus. Dann wird vom Portal eine Nachricht an den Router geschickt, der die Nachricht an den DPE Service weiterleitet. Beim DPE Service werden dann folgende Schritte durchgeführt:

- Der DPE Service erhält eine Nachricht vom Router.
- Anhand der Nachricht erkennt der Service, dass das übergebene Projekt komplett geladen werden soll.
- Der DPE Service verbindet sich mit der SQL Datenbank.
- Mit der übergebenen Projekt-ID wird das richtige Projekt gefunden und alle Daten, die für die Layoutplanung notwendig sind, aus der Datenbank ausgelesen.
- Das ausgelesene Projekt wird in eine Nachricht verpackt und an den Router geschickt.
- Der Router leitet die Nachricht an den Planungstisch Service und den Portal Service weiter.
- Im Planungstisch Service wird der Access Service aufgerufen, der sich mit der Accessdatenbank des Planungstisches verbindet und das aktuelle Projekt aus der Nachricht in die Accessdatenbank lädt.
- Im Portal Service wird die Nachricht in die Datenbank geschrieben, die auch vom Portal verwendet wird. Das Portal liest die Nachricht aus der Datenbank aus und zeigt auf der Oberfläche des Portals dem Benutzer an, dass die Layoutplanung im Planungstisch nun beginnen kann, da das von ihm ausgesuchte Projekt in die Accessdatenbank des Planungstisches geladen wurde.

Speichern der Daten nach Planungsende Wenn der Benutzer die Planung beendet hat betätigt er den Speichern-Button im Portal. Der Ablauf beim Speichern sieht dann wie folgt aus:

- Das Portal sendet eine Nachricht an den Router, der die Nachricht an den Planungstisch Service weiterleitet.
- Der Planungstisch Service ruft den Access Service auf, der sich mit der Accessdatenbank des Planungstisches verbindet und dort die Daten der fertigen Layoutplanung ausliest und an den Planungstisch Service zurückgibt.

- Der Planungstisch Service verpackt die Daten der Layoutplanung in eine Nachricht und schickt diese an den Router.
- Der DPE Service erhält eine Nachricht vom Router. In dieser Nachricht sind die Daten der Layoutplanung enthalten.
- Der DPE Service erkennt anhand der Nachricht, dass die Daten in der Datenbank gespeichert werden sollen.
- Der DPE Service verbindet sich mit der SQL Datenbank und aktualisiert die entsprechenden Tabellen mit den Daten der Layoutplanung.
- Der DPE Service erzeugt eine neue Nachricht, die an den Routing Service geschickt wird.
- Dieser leitet die Nachricht an den Portal Service weiter. Der Portal Service schreibt die Nachricht in die Datenbank. Dort holt sich das Portal die Nachricht über die Datenbank und aktualisiert die Oberfläche.

Datenbankschema

Das Datenbankschema, das erarbeitet wurde um die Struktur des DPE nachzubilden findet sich in Abbildung 3.12 wieder.
In diesem Datenbankschema wurde nur der Ressourcenbereich des DPE nachgebildet. Die zwei anderen Kernbereiche des DPE, Produkt und Prozess, mussten nicht beachtet werden, da momentan der Planungstisch das einzige System ist, das in der Architektur integriert wurde und dieser braucht nur Daten aus dem Ressourcenbereich.
Das Datenbankschema besteht aus folgenden Tabellen:

Projects: Diese Tabelle kann als *Oberknoten* angesehen werden. Hier stehen alle Projekte, die im DPE angelegt sind.

Factory: Alle angelegten Werke werden hier abgelegt.

Hall: In dieser Tabelle stehen alle verfügbaren Hallen, die im DPE angelegt sind.

Lines: Hier werden alle Linien einer Halle gespeichert.

Moduletype: Hier werden die verschiedenen Typen von Modulen gespeichert. Diese Daten sind für alle Module eines Typs gleich.

Modules: In dieser Tabelle werden die verwendeten Module mit dessen Lage in der Halle gespeichert.

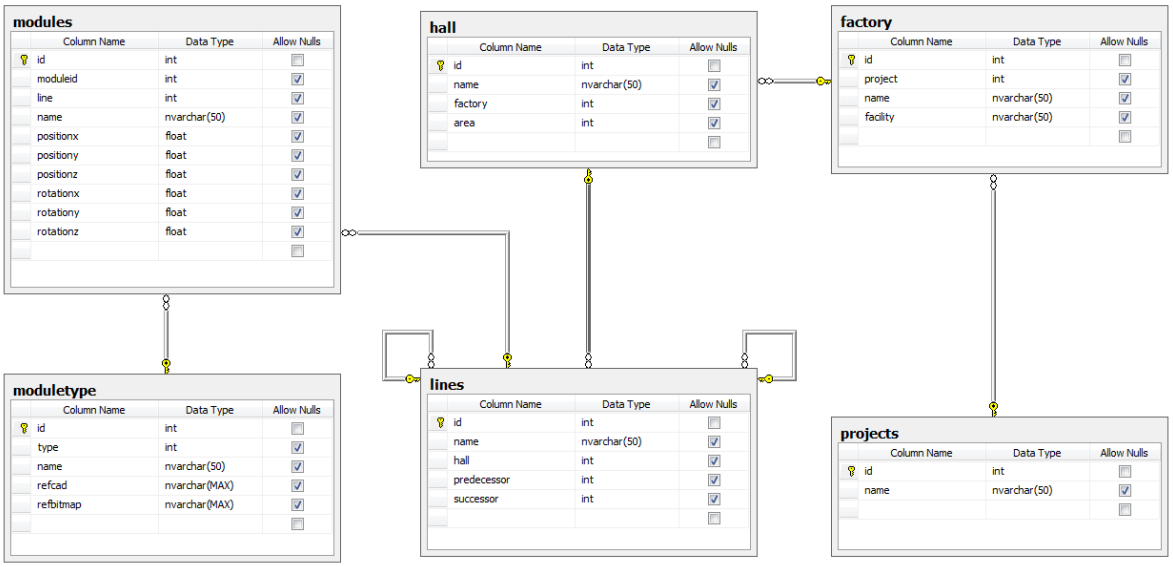

Abbildung 3.12.: Datenbankschema des DPE Services

3.3.6. Planungstisch Service

Der Planungstisch Service ist ein WCF Service. Er wird als Windows Dienst gehostet. Er besitzt eine eigene Warteschlange, die er ständig abhört. Seine Schnittstelle erwartet Nachrichten, die dem gemeinsamen XML Schema genügen.
Aufgabe dieses Services ist es Projekte in den Planungstisch zu laden oder nach Beendigung der Planung die Daten aus dem Planungstisch wieder auszulesen. Aber er greift dazu nicht direkt auf die Datenbank des Planungstisches zu.
Der Planungstisch Service ist ein WCF-Service, deswegen muss er auf *AnyCPU* kompiliert werden, das heißt für X86 und X64. Allerdings gibt es keine Datenbanktreiber für X64, die den Zugriff auf eine Accessdatenbank ermöglichen. Somit ist es nicht möglich von diesem Service direkt Daten in die Accessdatenbank zu schreiben oder sie wieder zu lesen (siehe Abschnitt 4.2.1).
Aufgrund dieser Tatsache dient der Planungstisch Service nur als Serviceendpunkt und ruft nach Erhalt einer Nachricht den Access Service auf. Der Access Service kann auf die Accessdatenbank des Planungstisches zugreifen, da er in X86 kompiliert ist und deswegen die angebotenen Datenbanktreiber benutzen kann.
Der Access Service stellt seine Methoden dem Planungstisch Service über .Net-Remoting zur Verfügung, so dass dieser somit indirekt auf die Accessdatenbank des Planungstisches zugreifen kann.

Ablauf

Je nach MessageType unterscheiden sich zwei Abläufe:

Nachricht ist vom Typ 1 (Projekt in der Datenbank des Planungstisches speichern):
Diese Nachricht wird wie folgt verarbeitet:

- Die Nachricht wird vom DPE Service über den Router an die Warteschlange des Planungstisch Services geschickt.
- Der PT Service hört die Warteschlange ab, nimmt die neue Nachricht entgegen und liest sie ein.
- Der PT Service erkennt, dass die Nachricht vom Typ 1 ist.
- Er bildet eine Instanz der Remote-Klasse *AccessConnection*.
- Aus der Remote-Klasse wird die Methode *UpdateDB* aufgerufen.
- Wenn die synchrone Methode ausgeführt wurde, wird die ursprüngliche Nachricht an die Warteschlange des Routers geschickt. Dieser leitet die Nachricht an den Portal Service weiter, der sie in die Datenbank schreibt.
- Dort wird sie vom Portal aus der Datenbank ausgelesen und die Oberfläche aktualisiert.

Nachricht ist vom Typ 2 (Projekt aus der Datenbank des Planungstisches auslesen):
Das Verarbeiten einer Nachricht von diesem Nachrichtentyp gliedert sich in folgende Schritte:

- Die Nachricht wird vom Portal an den Router geschickt, der sie an die Warteschlange des Planungstisch Services geschickt.
- Der PT Service hört die Warteschlange ab, nimmt die neue Nachricht entgegen und liest sie ein.
- Der PT Service erkennt, dass die Nachricht vom Typ 2 ist.
- Er bildet eine Instanz der Remote-Klasse *AccessConnection*.
- Aus der Remote-Klasse wird die Methode *GetDataFromDB* aufgerufen.
- Wenn die synchrone Methode ausgeführt wurde und das Projekt, das aus der Accessdatenbank ausgelesen wurde als Rückgabewert übermittelt wurde, dann verpackt PT Service das Projekt in eine Nachricht und sendet diese an die Warteschlange des Routing Service.
- Der Routing Service leitet die Nachricht an den DPE Service weiter, der das übergebene Projekt speichert und eine weitere Nachricht über den Router an den Portal Service sendet, dieser speichert die Nachricht in der Datenbank.
- Das Portal liest die Nachricht aus der Datenbank aus und aktualisiert die Oberfläche.

3.3.7. Access Service

Der Access Service dient als Brücke zwischen dem Planungstisch Service und dem Planungstisch. Der Access Service hat die Aufgabe die Projekte der Nachrichten, die beim Planungstisch Service ankommen, in die Accessdatenbank des Planungstisches zu laden. Außerdem ist er dafür verantwortlich die Ergebnisse nach Beendigung der Layoutplanung aus der Datenbank des Planungstisches wieder auszulesen, in eine Nachricht zu verpacken und diese an den Planungstisch Service weiterzuleiten.
Als Schnittstelle zwischen Planungstisch Service und Access Service dient eine Remoting-Klasse. Die Remoting-Klasse *AccessConnection* wird über eine URI zur Verfügung gestellt. Diese stellt die zwei Methoden *GetDataFromDB* und *UpdateDB* bereit, die beide synchron implementiert sind. *GetDataFromDB* liest die Daten der Layoutplanung aus der Datenbank des Planungstisches aus. Die Methode *UpdateDB* speichert ein aktuelles Projekt in der Datenbank des Planungstisches.

Ablauf

Der Ablauf gliedert sich in zwei verschiedene Teile. Zum einen das Speichern eines Projektes im Planungstisch und zum anderen das Auslesen eines Projektes, nach beendeter Layoutplanung, aus der Datenbank des Planungstisches heraus.

Projekt in der Datenbank des Planungstisches speichern Das Speichern eines Projektes in der Accessdatenbank des Planungstisches wird in folgende Schritte unterteilt:

- Der Planungstisch Service erzeugt eine Instanz der *AccessConnection-Klasse*.
- Der Planungstisch Service ruft die Methode *UpdateDB* der Klasse *AccessConnection* auf und übergibt dabei eine Instanz der Klasse *Project*, in der alle relevanten Informationen für den Planungstisch stehen.
- Der Access Service greift auf die Accessdatenbank des Planungstisches zu, löscht den Inhalt der Tabellen *ProjectObjects*, *ObjectAreas* und *ObjectPosition* und speichert den Inhalt des Objektes *Project* in die entsprechenden Tabellen.
- Nachdem die Methode *UpdateDB* ausgeführt wurde und die Tabellen aktualisiert hat fährt der Planungstisch Service wieder fort.
- Der Planungstisch Service schickt eine Nachricht an den Router, der diese an den Portal Service weiterleitet.
- Der Portal Service schreibt die Nachricht in die Datenbank. Danach liest das Portal die Nachricht wieder aus und aktualisiert die Oberfläche.

Daten nach Ende der Planung aus der Datenbank des Planungstisches auslesen Das Auslesen der Planungsdaten aus dem Planungstisch und das Erzeugen eines neuen Projektes wird in folgende Schritte unterteilt:

- Der Planungstisch Service erzeugt eine Instanz der *AccessConnection-Klasse*.
- Der Planungstisch Service ruft die Methode *GetDataFromDB* der Klasse *AccessConnection* auf.
- Die Methode *GetDataFromDB* greift auf die Accessdatenbank des Planungstisches zu und liest die relevanten Daten aus den Tabellen *ProjectObjects*, *ObjectAreas* und *ObjectPosition* aus.
- Nach dem Auslesen der Daten wird eine Instanz der Klasse *Project* erzeugt, in der die Daten aus den Tabellen gespeichert werden.
- Nach dem Erzeugen des Objekts *Project* wurde die Methode *GetDataFromDB* komplett ausgeführt und gibt dem Planungstisch Service das erzeugte Objekt als Rückgabewert zurück.
- Der Planungstisch Service schickt eine Nachricht an den Router, der diese an den DPE Service weiterleitet.
- Der DPE Service speichert die Daten der Layoutplanung und schickt dem Router eine Nachricht.
- Der Router schickt die Nachricht an den Portal Service weiter, der sie in die Datenbank schreibt.
- Die Nachricht wird vom Portal wieder aus der Datenbank ausgelesen und dem Benutzer das Ende der Layoutplanung angezeigt.

Datenbankschema

Das relevante Datenbankschema der Accessdatenbank des Planungstisches ist in Abbildung 3.13 zu sehen. Die wichtigsten vier Tabellen sind folgende:

AvailObj: In dieser Tabelle stehen alle Objekttypen. Sie wird nicht aktualisiert. Sie beinhaltet nur die Spalte *ID*, die in der Tabelle *ProjectObjects* als Fremdschlüssel auf die Spalte *ObjID* dient.

ProjectObjects: In dieser Tabelle stehen die einzelnen Objekte. Sie werden durch die Beziehung auf die Tabelle *AvailObj* genauer spezifiziert. Die beiden Spalten *ID* und *Position* sind identisch. Diese Tabelle wird beim Laden eines Projekts in den Planungstisch aktualisiert.

ObjectPosition: Diese Tabelle benutzt als Primärschlüssel die Spalte *ID*. Diese Spalte steht in Beziehung mit der Spalte *Position* aus der Tabelle *ProjectObjects*. In dieser Tabelle wird die genaue Lage der einzelnen Objekte festgehalten. Sie wird ebenfalls beim Laden eines Projekts in den Planungstisch aktualisiert.

ObjectAreas: Diese Tabelle benutzt als Primärschlüssel die Spalte *ID*. Diese Spalte steht in Beziehung mit der Spalte *Position* aus der Tabelle *ProjectObjects*. Diese Tabelle beschreibt die Größe der einzelnen Objekte. Die Tabelle dient nur zur internen Verarbeitung des Planungstisches. Damit dies fehlerfrei geschieht, muss die Tabelle bei jedem Laden eines Objektes in die Datenbank aktualisiert werden, dabei werden alle Spalten direkt auf 1 gesetzt.

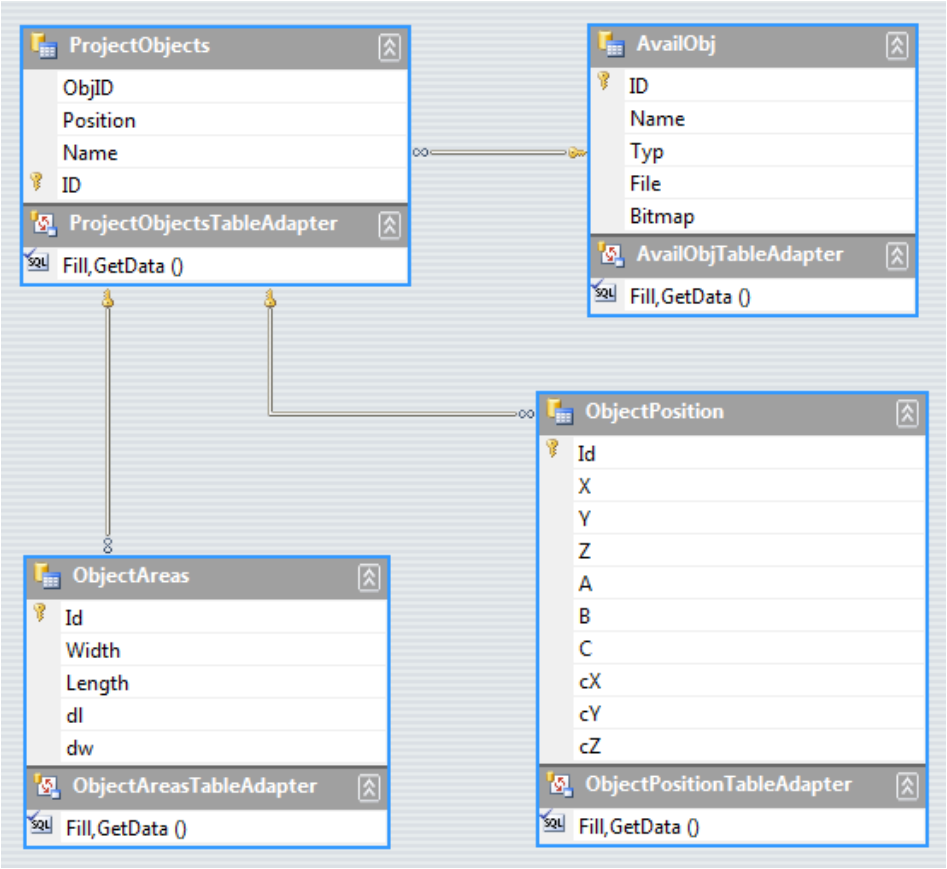

Abbildung 3.13.: Datenbankschema Access Service

3.3.8. Router

Der Router stellt die intelligente Komponente des Systems dar. In seiner Routingtabelle werden die Adressen aller beteiligten Systeme gespeichert und verwaltet. Durch diesen Aufbau müssen alle anderen Systeme nur den Endpunkt des Routers kennen.
Kommt bei dem Router eine Nachricht an, so wird diese analysiert und aufgrund des Inhaltes der Nachricht entschieden an welchen Service die Nachricht weitergeleitet werden muss. Das bestimmen der Zieladresse aufgrund des Inhalts einer Nachricht nennt man *Content Based Routing*.
Der Router ist als BPEL Prozess und eigenständigen WCF Service implementiert.

Wird die Routingkomponente als BPEL Prozess betrieben, so findet ein Aufruf an den Routing Service statt, dessen Routinglogik alle nötigen Endpunkte für die jeweilige Nachricht ermittelt. Der Aufruf der einzelnen Services geschieht dann allerdings wieder im Prozess. Wird der Router als Service betrieben, so findet die Verarbeitung der Nachricht, sowie das Aufrufen der einzelnen Komponenten direkt im Service statt. Laut Definition eines ESBs muss das Routen von dem ESB übernommen werden, deshalb wurde der BPEL-Prozess erstellt. Im momentanen Betrieb wird der WCF-Service verwendet, der aber auf dem gleichen System wie der ESB ausgeführt wird.
In Abschnitt 3.1.1 wird der BPEL Prozess genauer beschrieben. In diesem Teil werden nur die Routinglogik und das Verhalten des Routing Services beschrieben.

Routinglogik

Die Routinglogik bestimmt das Ziel, bzw. die Ziele einer Nachricht aufgrund dem Inhalt der Nachricht. Dies findet anhand von XPATH-Ausdrücken statt, die in einer Datenbank, siehe Abschnitt 3.14, hinterlegt sind.
Beispiel für einen XPATH-Ausdruck:

```
/*/CommonInfos[MessageIdregistered="false" or MessageId=""]
```

Dieser Beispiel-Ausdruck überprüft, ob die ankommende Nachricht bereits in der Message Registry registriert ist oder nicht.
Die Routinglogik ist ebenfalls dafür verantwortlich, dass die Nachrichten an die zwei Services Messageregistration und Messageflowregistration weitergeleitet und dort registriert werden.

Ablauf Die Routinglogik führt folgende Schritte durch:

- Die Nachricht wird vom Routing Service entgegengenommen und an die Routinglogik weiter gegeben.
- Aus der Tabelle *Messages* werden nacheinander die XPATH-Ausdrücke gelesen und an der XML-Datei der eingehenden Nachricht ausgewertet.
- Enthält die XML-Datei den gesuchten XPATH-Ausdruck, so wird die Suche abgebrochen und im nächsten Programmschritt weiter gemacht, ansonsten wird der nächste XPATH-Ausdruck aus der Tabelle geholt und dieser untersucht.
- War das Auswerten erfolgreich, so werden aus der Tabelle *Receivers* alle Zeilen ausgelesen, dessen Message_ID mit der ID der Zeile des gefundenen XPATH-Ausdrucks übereinstimmen.
- Anhand der Service_ID, die aus der Tabelle *Messages* stammt, kann nun der Uniform Resource Locator (URL) aus der Tabelle *Services* ausgelesen werden.
- Nun werden alle URLs, in der richtigen Reihenfolge (durch order in Tabelle *Receivers* festgelegt) in die Nachricht geschrieben.

- Wenn das Erstellen der Nachricht beendet ist wird noch überprüft, ob die Spalte routerinternal des gefunden XPATH-Ausdrucks in der Tabelle *Messages* auf *true* oder *false* gesetzt war.
- Wenn *true* gesetzt war, dann wird sofort die erste URL der Liste aufgerufen und anschließend gelöscht, wenn nicht, dann wird der BPEL Prozess aufgerufen.

Routing Service

Der Routing Service ist ein WCF Service. Er wird als Windows Dienst gehostet. Er besitzt eine eigene Warteschlange, die er ständig abhört. Seine Schnittstelle erwartet Nachrichten, die dem gemeinsamen XML Schema genügen.
Aufgabe des Services ist das Empfangen und Weiterleiten von Nachrichten an die richtige Adresse.

Ablauf Die folgenden Schritte führt der Routing Service aus, wenn er eine Nachricht erhält:

- Der Routing Service empfängt eine Nachricht.
- Er sendet die Nachricht weiter an die Routinglogik, von der die Nachricht ausgewertet wird und die entsprechenden Endpunkte der Nachricht hinzugefügt werden.
- Die Routinglogik entscheidet dann, ob ein externer Service im Anschluss aufgerufen werden muss.

Datenbankaufbau

Abbildung 3.14 zeigt das Datenbankschema des Routers. Mit Hilfe dieser Tabellen entscheidet die Routinglogik an welche Services die Nachrichten geschickt werden müssen. Es werden folgende Tabellen benötigt:

Messages: Hier stehen alle Nachrichten, die der Router empfangen kann. Diese werden durch die XPATH-Ausdrücke klassifiziert.

Services: In dieser Tabelle stehen alle vorhanden Services mit ihren Beschreibungen und Endpunkten. Wird ein Service auf einen anderen Rechner verlagert, so muss nur die URL angepasst werden.

Receivers: Diese Tabelle stellt das Bindeglied zwischen *Messages* und *Services* dar. Hier wird spezifiziert an welche Systeme eine Nachricht weitergeleitet werden muss, wenn die Nachricht von einem bestimmten Nachrichtentyp ist.

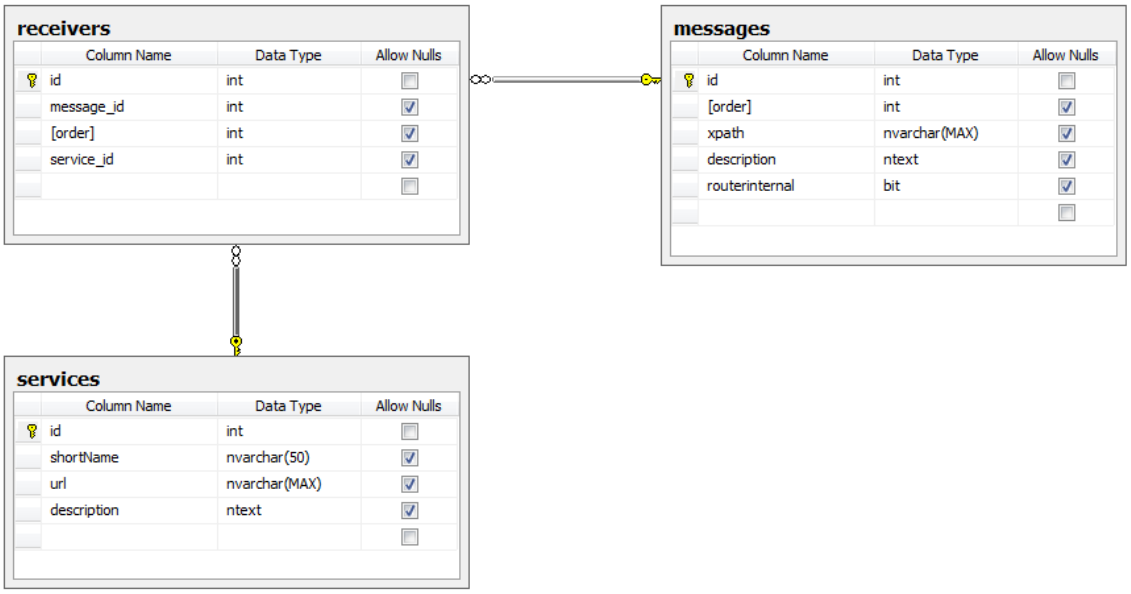

Abbildung 3.14.: Datenbankschema Router

3.3.9. Messageregistration Service

Der Messageregistration Service ist ein WCF Service. Er wird als Windows Dienst gehostet. Er besitzt eine eigene Warteschlange, die er ständig abhört. Seine Schnittstelle erwartet Nachrichten, die dem gemeinsamen XML Schema genügen.
Aufgabe des Services ist die Registrierung der Nachrichten, damit es später möglich ist den Nachrichtenverlauf zu verfolgen. Dies erleichtert es, eventuell auftretende Fehler zu suchen.

Ablauf

Der Ablauf gliedert sich in folgende Schritte:

- Der Routing Service schickt eine Nachricht an die Warteschlange des Messageregistration Service.
- Da der Messageregistration Service die Warteschlange ständig abhört, erkennt er die ankommende Nachricht und nimmt sie entgegen.
- Der Service erzeugt eine eindeutige Messsage-ID.
- Die Message-ID wird in die Nachricht eingetragen.
- Die Nachricht wird in der Datenbank abgespeichert.

– Der Messageregistration Service schickt die modifizierte Nachricht an die Warteschlange des Routers.

Datenbankaufbau

Die Datenbank des Messageregistration Service besteht lediglich aus einer Tabelle, sie heißt *Messages*. In dieser Tabelle werden alle Nachrichten gespeichert, die das System zum ersten Mal durchlaufen. Das Datenbankschema dieses Services ist in Abbildung 3.15 dargestellt.

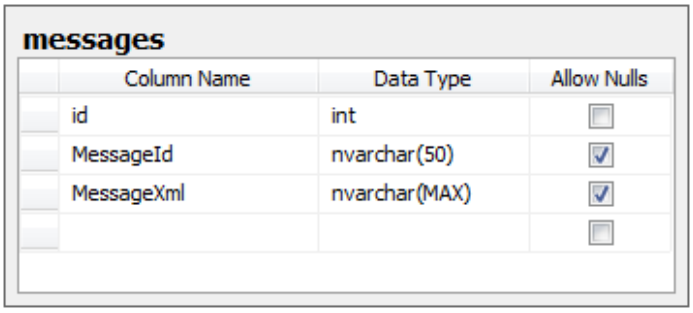

Abbildung 3.15.: Datenbankschema Messageregistration Service

3.3.10. Messageflowregistration Service

Der Messageflowregistration Service ist ein WCF Service. Er wird als Windows Dienst gehostet. Er besitzt eine eigene Warteschlange, die er ständig abhört. Seine Schnittstelle erwartet Nachrichten, die dem gemeinsamen XML Schema genügen.
Die Aufgabe des Messageflowregistration Service ist es einen Nachrichtenfluss zu generieren. Das heißt, Nachrichten, die zusammen gehören, mit gemeinsamen IDs zu versehen. Ist die Nachricht in noch keinem Nachrichtenfluss enthalten, so wird eine eindeutige ID, die MessageFlow-ID, erstellt.
Dieser Service wurde ausschließlich zur Fehlersuche implementiert. Um den Ursprung einen aufgetretenen Fehlers zu bestimmen genügt es den Nachrichtenfluss zu verfolgen. Ein Fehler trat bei dem System auf, bei dem der Nachrichtenfluss endet.

Ablauf

Der Ablauf gliedert sich in folgende Schritte:

– Der Router schickt eine Nachricht an die Warteschlange des Messageflowregistration Service.

- Da der Messageflowregistration Service die Warteschlange ständig abhört, erkennt er die ankommende Nachricht und nimmt sie entgegen.
- Der Service überprüft, ob die Nachricht bereits in einem Nachrichtenfluss enthalten ist. Falls ja, dann ordnet er die Nachricht diesem Fluss zu, falls nein, wird ein neuer Fluss mit einer eindeutigen MessageFlow-ID erzeugt.
- Die die MessageFlow-ID wird in die Nachricht eingetragen.
- Die Nachricht wird in der Datenbank abgespeichert.
- Der Messageflowregistration Service schickt die modifizierte Nachricht an die Warteschlange des Routers.

Datenbankaufbau

Abbildung 3.16 zeigt das Datenbankschema des Messageflowregistration Service. Die Datenbank des Messageflowregistration Service besteht aus zwei Tabellen:

Messages: Die Tabelle *Messages* enthält alle Nachrichten, die zu einem Nachrichtenfluss gehören.

MessageFlows: Die Tabelle speichert alle Nachrichtenflüsse, die bereits registriert wurden. Neue Nachrichtenflüsse werden auch in diese Tabelle eingefügt. Es wird jeweils der Startzeitpunkt des Flusses mit abgespeichert.

3.4. Implementierung

3.4.1. Services

Alle Services dieser Arbeit sind Windows Communication Services. Sie besitzen alle das gleiche Interface. Dieses wurde einmal erstellt und dann in jedem Service wieder verwendet. Die Logik der einzelnen Services wurde separat in den jeweiligen Services implementiert. Für den Nachrichtenaustausch wurden Klassen erstellt, die das XML-Schema aus Abschnitt 3.2 eins zu eins repräsentieren. Diese Klassen, sowie weitere Hilfsklassen, wie zum Beispiel zum Aufrufen der Message Queues, zum Serialisieren und Deserialisieren aller Klassen, zum Schreiben und Lesen von XML-Files und zum Umrechnen von Werten befinden sich in der *Common.dll* Bibliothek.

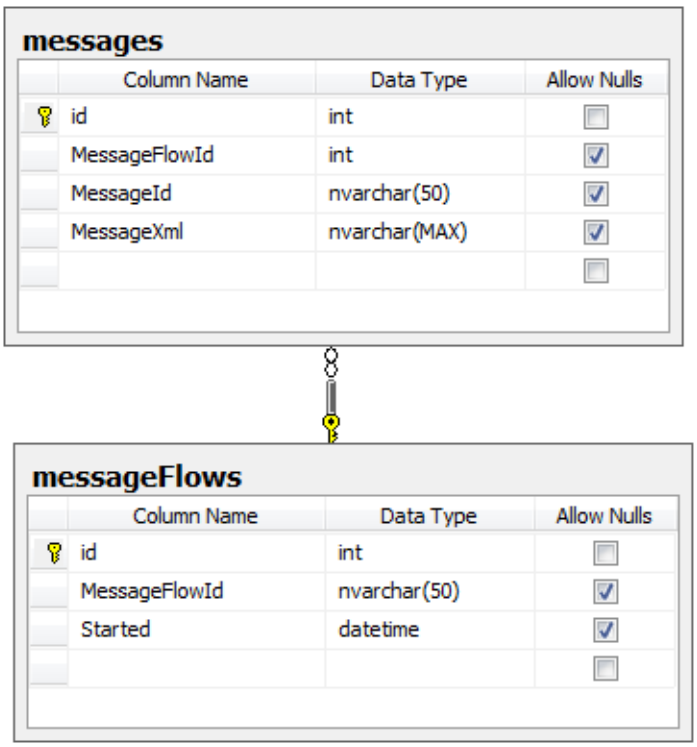

Abbildung 3.16.: Datenbankschema Messageflowregistration Service

Basisklasse

Außerdem wurde eine Basisklasse für das Abarbeiten der verschiedenen Nachrichtentypen entwickelt. Am Anfang wird immer die öffentliche Methode *ProcessMessage* aufgerufen. In dieser wird dann je nach Nachrichtentyp entschieden, welche der folgenden Methoden aufgerufen wird:

- On_ProcessAllMessages
- On_GetAllProjects
- On_GetDataFromDpe
- On_GetDataFromPt

Die Methode *On_ProcessAllMessages* wird immer aufgerufen, hier können allgemein gültige Arbeitsschritte erledigt werden. Über den Aufruf der restlichen Methoden entscheidet der MessageType.

Aufruf der Message Queues

Jeder Service hat einen Endpunkt mit dessen Hilfe sich die Namen der Message Queues, die der Service abhört, herausfinden lassen. Mitgelieferte Klassen des .Net-Frameworks stellen Hilfsmechanismen zur Verfügung, diese Message Queues aufzurufen und Nachrichten an sie zu senden.
Der Aufruf einer solchen Message Queue und damit das Aufrufen eines Services erfolgt in folgenden Schritten:

- Mit Hilfe der URI des Services wird eine Proxyklasse erzeugt.
- Aus dieser Proxyklasse erhält man den Namen der Message Queue, die der Service abhört.
- Es wird ein Binding erzeugt.
- Mit Binding und der Channelfactory wird ein Servicechannel erzeugt.
- Über diesen Servicechannel lassen sich Nachrichten senden, die dem gemeinsamen Schema entsprechen.
- Damit jede Nachricht diesem gemeinsamen Schema entspricht, werden sie vor dem Versenden in dieses Format serialisiert.

Konfigurationsdatei

Um feste Konfigurationen einer Anwendung abzulegen werden im .Net-Framework sogenannte Konfigurationsdateien verwendet. Das sind üblicherweise XML-Dateien, aber die Endung *.config* besitzen.
In diesen Dateien sind die Interfaces der Services beschrieben. Außerdem werden *ConnectionStrings* für Datenbankverbindungen oder die URI des Routers gespeichert.
Diese Dateien werden bei jedem Start des Services eingelesen, aber können vor dem Einlesen noch verändert werden. Das bedeutet, dass zum Beispiel der Datenbankbenutzer oder die URI des Routers nachträglich noch bearbeitet werden können ohne den Service neu kompilieren zu müssen.

3.4.2. Portal

Das in Abschnitt 3.3.1 vorgestellte Portal wurde in ASP.NET programmiert. Diese Technologie dient als Grundgerüst zum Erstellen dynamischer Webseiten. Es gibt dabei zwei Elemente, auf dessen Grundlage alle Seiten des Portals erstellt wurden:

Masterpage

Die Masterpage dient allen angezeigten Webseiten als Vorlage. Sie besteht aus zwei Teilen. Am oberen Bildschirmrand wird dauerhaft das Titelbild des Portals angezeigt und im unteren Teil ist ein Platzhalter definiert, der von den einzelnen Webseiten passend zum aktuellen Status des Workflows gefüllt wird.
Außerdem wird eine Cascading Style Sheets (CSS) Datei, die *StyleSheet.css*, eingebunden. In dieser Datei werden die verschiedenen Styles definiert:

- body
- top
- left
- input
- right
- center

Die Verwendung von CSS dient dem einheitlichen Aussehen aller verwendeten Seiten.

Session-Handling

ASP.NET hat ein integriertes Session-Handling. Das heißt, es bietet die Möglichkeit einen Status über mehrere Seiten hinweg zu transportieren. In diesem *Session-Objekt* kann man verschiedene, auch selbst definierte, Objekte speichern und auf der nächsten Seite wieder darauf zugreifen. So ist es möglich zu überprüfen, ob ein Benutzer angemeldet ist oder nicht, sonst müsste auf jeder Seite erneut der Anmeldebildschirm gezeigt werden.
In der eigens für diesen Zweck definierten Klasse *Users* werden alle Benutzer gespeichert, die momentan am System angemeldet sind.

3.4.3. BPEL Systeme

In dieser Arbeit werden zwei BPEL Prozesse verwendet, zum einen Routing Prozess und zum anderen den Planungs Prozess. Beide werden direkt auf der BPEL Engine des Open ESB ausgeführt.

Routing Prozess

Der Routing Prozess hat einen Endpunkt, über den er von anderen Anwendungen aufgerufen werden kann. Außerdem hat er eine Schnittstelle, über die er fremde Services aufrufen kann. Das integrierte JBI-Module und die zwei WSDL-Schnittstellen finden sich in Abbildung 3.17.

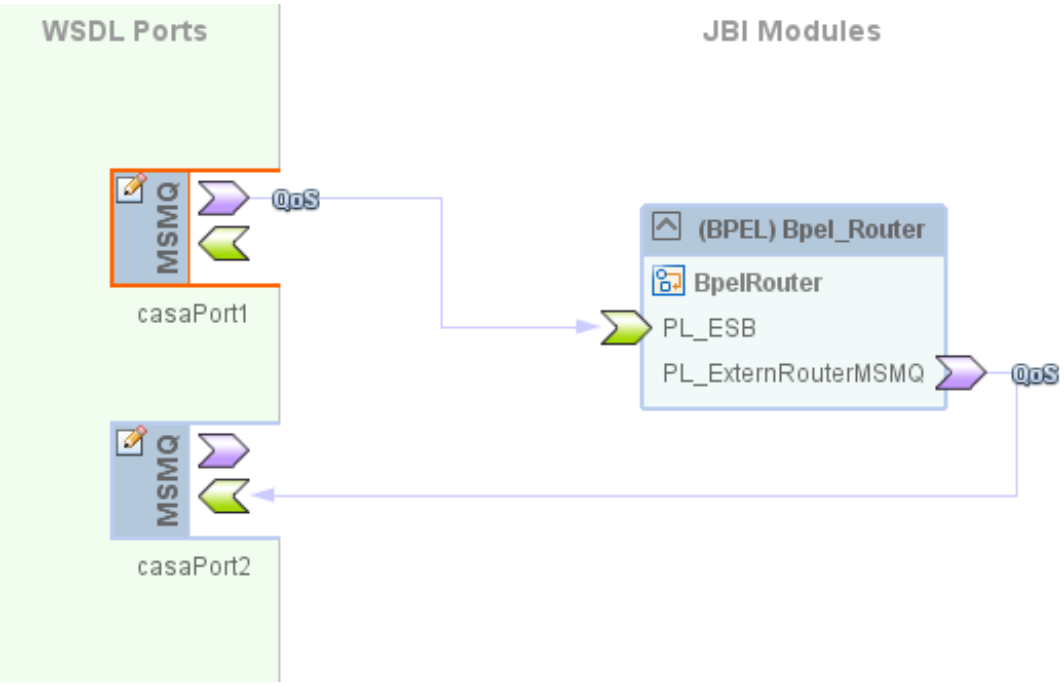

Abbildung 3.17.: Routing Workflow mit Endpunkte

Planungs Prozess

Der Planungs Prozess besitzt ebenfalls einen Endpunkt, über den er Nachrichten von anderen Anwendungen entgegen nimmt. Zusätzlich besitzt der Prozess eine Schnittstelle, über die er andere Anwendungen aufrufen kann. Dabei werden alle Nachrichten, die im Prozess ankommen, vom Router geschickt. Alle Nachrichten, die der Prozess selbst verschickt, werden an den Router geschickt, der die Endpunkte der entsprechenden Services bestimmt und die Nachrichten an diese Adressen weiterleitet.
Das JBI-Module, in dem der Prozess gehostet wird, ist ähnlich zu dem in Abbildung 3.17. Einziger Unterschied ist der integrierte Prozess, der aus dem Planungs Prozess besteht.

3.4.4. Datenbankzugriffe

In allen Services und in dem Portal werden Datenbankzugriffe benötigt. Da alle diese Anwendungen in C# implementiert werden kann der Zugriff mit ADO.NET-Komponenten realisiert werden.

DataSet und DataTable

DataSets sind Objekte, die im Arbeitsspeicher gehalten werden. Es wird eine Datenbankverbindung hergestellt und die entsprechenden Daten geladen. Dabei wird jede Tabelle, die geladen werden soll in einer eigenen DataTable gespeichert. Das Zusammenfassen von mehreren DataTables nennt man ein DataSet.
Eine DataTable ist das Abbild einer Datenbanktabelle. Um auf den Inhalt einer Zeile zugreifen zu können, muss diese erst identifiziert werden. Ist dies geschehen, dann kann auf den Inhalt der einzelnen Spalten zugegriffen werden [msd].

DataAdapter

DataAdpater stellen die Verbindung von Datenquelle und dem DataSet dar. So werden die vier wesentlichen SQL-Befehle von den DataAdaptern ausgeführt, die bereits beim Erzeugen eines DataAdapters zur Verfügung gestellt werden [msd]. Die vier Befehle lauten:

- Select
- Update
- Insert
- Delete

Alle in 3.3 vorgestellten Systeme, die eine Datenbankverbindung benutzen, verwenden DataSets. Dabei werden für alle Tabellen, die ausgelesen oder bearbeitet werden müssen, DataTables und DataAdapter erstellt und pro Anwendung zu einem DataSet zusammengefasst. Aus diesem Grund hat reicht es aus pro System nur eine Datenbankverbindung aufzubauen, die gehalten werden muss.

4. Bewertung

In diesem Kapitel werden mögliche Abläufe bei der Ausführung der Layoutplanung beschrieben. Außerdem werden die Probleme, die bei der Umsetzung aufgetreten, sind behandelt. Ein weiterer Punkte, der besprochen wird, ist die Wartbarkeit des Systems. Bevor am Ende des Kapitels noch ein Vergleich zwischen den gestellten Problemen, den angestrebten Zielen und dem Resultat der Arbeit aufgestellt wird, wird das Thema dieser Arbeit mit anderen Arbeiten verglichen, die ähnliche Themen behandelt haben.

4.1. Betrieb

In diesem Abschnitt werden verschiedene Szenarien beschrieben, die während der Ausführung auftreten können. Dabei wird neben dem Gesamtsystem auch das Verhalten der einzelnen Komponenten untersucht.

4.1.1. Szenario: Durchführung einer Layoutplanung

Der Ablauf einer Layoutplanung ist in Abbildung 4.1 zu sehen.
Der Benutzer startet die Layoutplanung. Dazu lädt der DPE alle Projekte, welche dann im Portal angezeigt werden. Anschließend entscheidet sich der Benutzer für ein Projekt, das in den Planungstisch geladen werden soll und anschließend dort zur Bearbeitung bereit seht. Dabei wird das Projekt aus dem DPE geladen und in der Accessdatenbank des Planungstisches gespeichert. Nun bearbeitet der Benutzer das Projekt im Planungstisch. Nachdem die Layoutplanung abgeschlossen ist, veranlasst der Benutzer das System seine Daten wieder im DPE zu speichern.

4.1.2. Szenario: Layoutplanung mit Unterbrechung

In Abbildung 4.2 wird beschrieben was passiert, wenn die Planung unterbrochen wird und später fortgesetzt werden soll.
Es kann mehrere dafür Gründe geben, dass eine Layoutplanung unterbrochen wird. Die erste Möglichkeit ist, dass sich der Benutzer selbst ausloggt mit der dafür vorgesehenen Schaltfläche auf der Portaloberfläche. Der andere Fall tritt ein, wenn der Benutzer auf dem Portal zu lange inaktiv ist, dann wird er vom System ausgeloggt.
Immer wenn der Benutzer die Planung unterbrochen hat und sich wieder anmeldet, dann wird überprüft, ob noch ein Workflow aktiv ist. Falls dies der Fall ist, dann hat der Benutzer

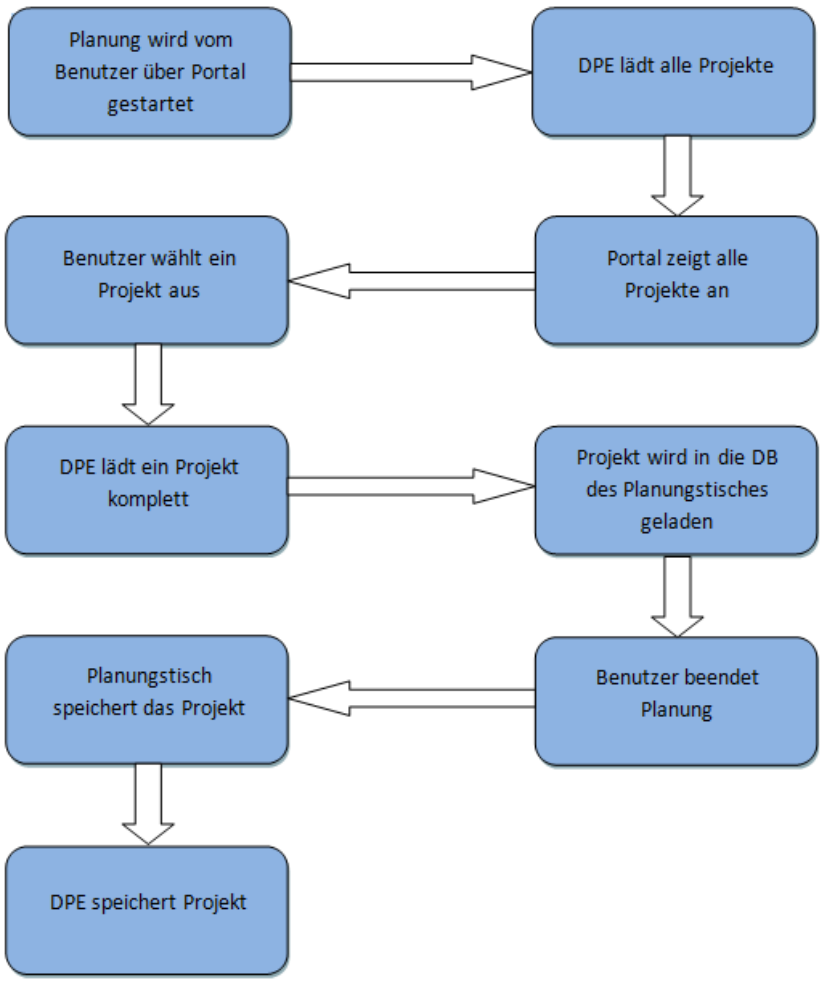

Abbildung 4.1.: Durchführung einer Layoutplanung

die Möglichkeit den Workflow, der sich im Wartezustand befindet, fortzuführen oder einen neuen Workflow zu starten.
Entscheidet sich der Benutzer dafür einen neuen Workflow zu starten und der Benutzer war bereits dabei ein Projekt am Planungstisch zu bearbeiten, dann gehen die Änderungen des Projektes verloren. Dies liegt daran, dass jedes mal beim Laden eines Projektes in die Datenbank des Planungstisches alle Tabellen, die aktualisiert werden, vorher komplett geleert werden.
Entscheidet sich der Benutzer allerdings den wartenden Workflow fortzuführen, so wird überprüft in welchem Status sich der Workflow befindet und genau an dieser Stelle wird der Workflow fortgesetzt. Um den aktuellen Status des Workflows festzustellen wird die letzte Nachricht aus der Tabelle *Currentessage* der Datenbank des Portals gelesen. Dies war die letzte Nachricht, die vom Portal aus gesendet wurde, somit ist dies auch der aktuelle Status des gesamten Workflows.

Abbildung 4.2 dient nur als Beispiel, es ist unerheblich in welchem Arbeitsschritt die Layoutplanung unterbrochen wird, es kann immer die entsprechende Stelle herausgefunden werden und die Planung von diesem Schritt aus fortgesetzt werden.

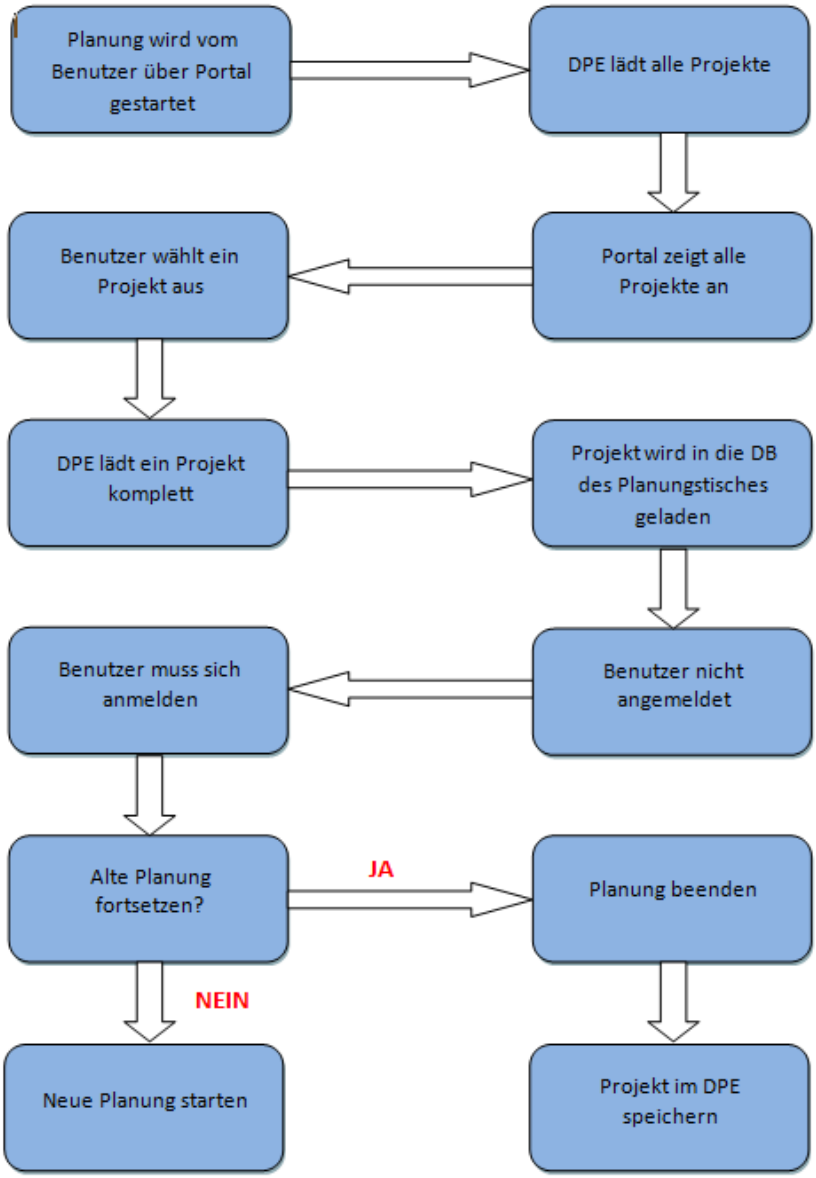

Abbildung 4.2.: Layoutplanung mit Unterbrechung

4.1.3. Szenario: System fällt aus

In Abbildung 4.3 wird beispielhaft gezeigt was passiert, wenn ein System zeitweise nicht verfügbar ist.
Jedes System hat eine eigene Warteschlange, auf die Nachrichten gelegt werden, wenn ein System aufgerufen werden soll. Kommt eine Nachricht auf der Warteschlange an und das System ist verfügbar, so werden die Nachrichten von den Systemen entgegengenommen und verarbeitet. Alle Systeme wurden so konzipiert, dass sie dauernd ihre Warteschlangen abhören und die Nachrichten entgegennehmen.
Fällt ein System zeitweise aus, so werden die Nachrichten so lange in den Warteschlangen gehalten, bis das System wieder verfügbar ist und die Nachrichten einliest oder sie manuell aus den Warteschlangen gelöscht werden.
Für den Ablauf der Layoutplanung und das Verhalten der anderen Systeme stellt es kein Problem dar, wenn ein System zeitweise nicht verfügbar ist. Da der komplette Nachrichtenaustausch asynchron verläuft kann es auch keine Folgefehler, wie etwa das Überschreiten eventueller Timeouts, nach sich ziehen. Die einzige Beeinträchtigung ist, dass die Planung so lange verzögert wird, bis das ausgefallene System wieder in Betrieb ist.
Da alle Services mit Warteschlangen ausgerüstet sind, ist es unerheblich welches System kurzzeitig nicht verfügbar ist. Ein weiterer Vorteil von Warteschlangen ist, dass sie die lose Kopplung von Services unterstützt. Da der Endpunkt, die Message Queue immer gleich bleibt, können die Systeme, welche die Warteschlange abhören, ausgetauscht werden.

4.2. Probleme

In diesem Abschnitt werden die Probleme besprochen, die bei der Implementierung entstanden sind.

4.2.1. Zugriff auf die Accessdatenbank

Problem

Ein Problem war der Zugriff auf die Accessdatenbank des Planungstisches. Anfangs wurde versucht über den Planungstisch Service auf die Accessdatenbank zuzugreifen, was im Prinzip auch kein Problem darstellen sollte, da der Service in C# implementiert wurde und so über ADO.NET auf Accessdatenbank zugegriffen werden kann.
Doch alle C#-Projekte, die eine WCF-Library benutzen, müssen auf *AnyCPU* kompiliert werden. Das heißt die Kompilierung läuft für X64 und X86. Doch leider gibt es bis heute noch keine Treiber für X64, die auf Accessdatenbanken zugreifen können.
Also war folgendes Problem gegeben: Entweder konnte der Service keine Nachrichten empfangen, da er nicht richtig lief oder es konnte nicht auf die Accessdatenbank zugegriffen werden.

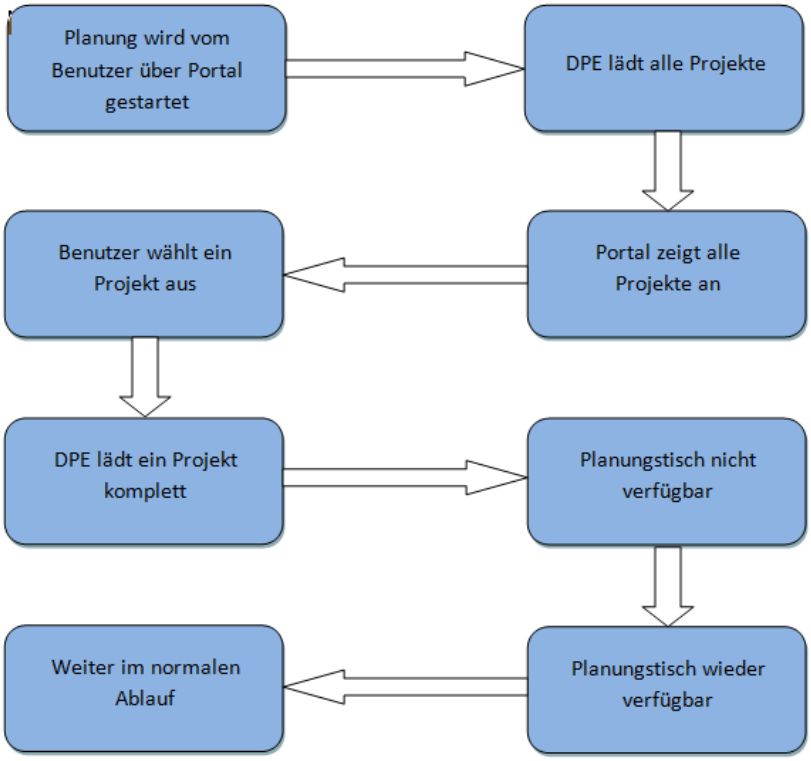

Abbildung 4.3.: System fällt aus

Lösung

Das Problem wurde gelöst, indem ein zusätzlicher Service erstellt wurde. Abbildung 4.4 zeigt den Aufbau, der verwendet wurde um das Problem der fehlenden Treiber für Accessdatenbanken zu lösen.
Es wurde ein zusätzlicher Service, der *Access Service* implementiert. Dieser Service wird nur in X86 kompiliert und kann somit auf die Accessdatenbank zugreifen.
Allerdings musste noch eine Möglichkeit gefunden werden um auf diesen Service zuzugreifen. Da der Planungstisch Service auf *AnyCPU* kompiliert wurde und der Access Service nur auf X86, war es nicht möglich die Referenz des Access Services in das Projekt des Planungstisch Services einzubinden, da sonst der Planungstisch Service wieder in X86 ausgeführt worden wär um zu der eingebundenen Referenz kompatibel zu sein. Aus diesem Grund wurde in der *Common.dll*, die ebenfalls auf *AnyCPU* kompiliert wurde, ein Interface Namens *IAccessDB* erstellt. Dieses Interface stellt die zwei Methoden *GetDataFromDB* und *UpdateDB* zur Verfügung. Diese werden vom Access Service in der Klasse *AccessConnection* implementiert und greifen bei ihrer Ausführung jeweils auf die Accessdatenbank zu. Während *GetDataFromDB* das aktuelle Projekt aus der Datenbank des Planungstisches ausliest und zurück gibt, schreibt *UpdateDB* das übergebene Projekt in die selbige.
Um die Implementierung des Interfaces dem Planungstisch Service zur Verfügung zu stellen

veröffentlicht der Access Service die Klasse *AccessConnection* unter der lokalen Adresse *tcp://localhost:19001/AccessDB.rem*.
Wenn der Planungstisch Service gestartet wird, wird ein Proxyobjekt dieser Adresse erstellt. Da dem Planungstisch Service das Interface *IAccessDB* ebenfalls bekannt ist, kann er nun auf das Proxyobjekt zugreifen. Immer wenn vom Planungstisch Service auf die Accessdatenbank zugegriffen werden muss, wird das Proxyobjekt verwendet und eine der zwei Methoden aufgerufen.
Somit wurde erreicht, dass ein WCF-Service verwendet werden kann und trotzdem die Möglichkeit besteht auf die Accessdatenbank zuzugreifen. Da beide Services und die Datenbank auf dem selben Rechner ausgeführt werden und nicht zu erwarten ist, dass die Objekte, die über das Remoting ausgetauscht werden, groß sind, stellt dies eine gute Lösung dar.

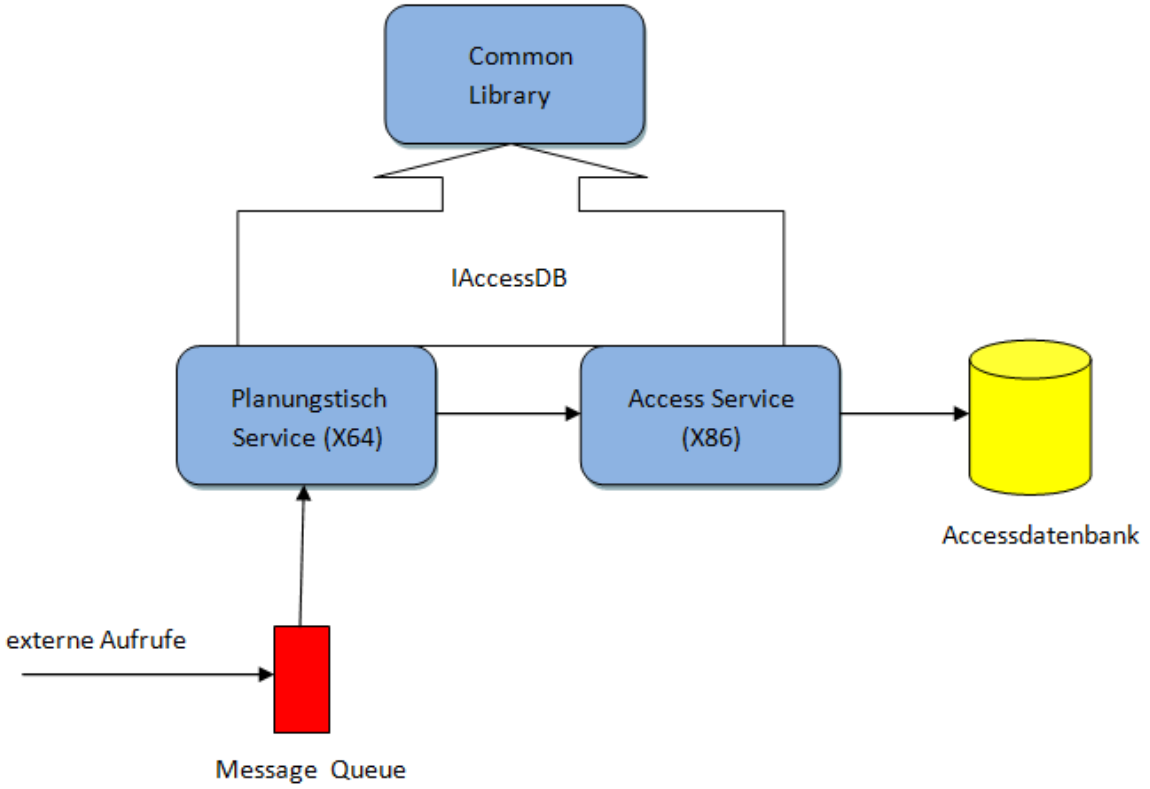

Abbildung 4.4.: Aufbau zur Lösung des Datenbankzugriffs

4.2.2. Schwierige Kommunikation zwischen Open ESB und WCF Services

Teilweise war es schwierig die Kommunikation zwischen Open ESB und den WCF Services zu ermöglichen. Dies sollte eigentlich durch das SOA Prinzip verhindert werden, doch es kam dennoch zu Problemen.
Es war nicht möglich Serviceaufrufe ohne ein konkretes Binding anzugeben. Dies hingegen

war beim Implementieren der WCF Services kein Problem. So wurde ein standardisiertes Binding verwendet, das sowohl beim Open ESB, als auch bei den WCF Services funktionierte.

4.2.3. Fehlende Schnittstelle beim DPE

Problem

Der DPE besitzt keine Schnittstelle, auf die man von anderen Anwendungen aus zugreifen kann. Doch um den DPE in die Systemlandschaft integrieren zu können muss er von anderen Anwendungen aus erreichbar sein.

Lösung

Dieses Problem wurde dadurch gelöst, dass einmalig im DPE ein VBA-Skript gestartet wird, das in einer Endlosschleife läuft. Das Skript versucht ständig Dateien einzulesen, die an einem fest vorgegebenen Pfad hinterlegt sind. Diese Dateien beinhalten Nachrichten, die im DPE verarbeitet werden.

4.2.4. Kommunikation zwischen VBA und C#

Problem

Es ist mit VBA 6.0, was im DPE verwendet wird, nicht möglich direkt WCF Services aufzurufen.

Lösung

Um das Problem zu lösen wurde eine Klasse *VBAInterface* in C# erstellt. In dem gleichnamigen Projekt wurde wieder die Bibliothek *Common.dll* eingebunden. Mit Hilfe dieser Klasse und der Referenz auf die Bibliothek war es möglich Services aufzurufen.
Nun musste noch die Möglichkeit geschaffen werden von dem VBA-Skript auf die Klasse *VBAInterface* und die Methode *CallService* zugreifen zu können. Um dies zu realisieren wurde die Klasse *VBAInterface* als COM-Klasse veröffentlicht. Da COM-Referenzen in VBA-Projekte eingebunden werden können, konnte die Referenz auf diese Klasse in das VBA-Projekt eingebunden werden und ein Zugriff darauf stattfinden.

4.3. Wartbarkeit

Das System weißt einen hohen Grad an Wartbarkeit auf, was vor allem an der losen Kopplung der einzelnen Systeme liegt. Durch den Einsatz von SOA und die Verwendung von Message Queues wurde eine lose Kopplung erreicht.
Es ist problemlos möglich bestimmte Services auszutauschen, da bei den Services der Endpunkt von der Implementierung getrennt wurde. Wird nun ein System ausgetauscht, so wird nur die Implementierung des Services ausgetauscht, der Endpunkt hingegen bleibt der gleiche, da die Message Queue weiterhin vorhanden ist.
Das Verschieben eines Services auf einen anderen Rechner stellt auch kein Hindernis dar, da der Router der einzige Service im Gesamtsystem ist, dessen Service Endpunkt den anderen Services bekannt ist. Im Gegensatz dazu stehen alle Service Endpunkte in der Routingtabelle des Routers. Wird ein Service auf einem neuen Rechner ausgeführt, so muss lediglich der Service Endpunkt in der Routingtabelle geändert werden. Gleich verhält es sich beim Hinzufügen eines Services. In der Routingtabelle muss ein neuer Service mit dem entsprechenden Endpunkt angelegt und im Nachrichtenfluss an der richten Stelle eingetragen werden.
Durch die Verwendung eines gemeinsamen Nachrichtenmodells, das an einem zentralen Punkte liegt, der *Common.dll*, kann dies ohne großen Aufwand geändert werden. Da sich eine einzige Änderung am Nachrichtenmodell auf das Nachrichtenmodell aller Systeme auswirkt, entsteht nur ein geringer Wartungsaufwand.

4.4. Verwandte Arbeiten

In diesem Abschnitt werden Arbeiten vorgestellt, die sich mit Themen beschäftigen, die ähnlich zu dieser Arbeit sind. Es werden die Gemeinsamkeiten und Unterschiede herausgearbeitet.

4.4.1. Event-driven Production Rescheduling

In der Arbeit von Frank Ruthardt *Event-driven Production Rescheduling* [Rut10] wurden verschiedene Systeme der physischen Lernfabrik miteinander verbunden. Zu den Systemen zählen unter anderem der Leitrechner für die Fabriksteuerung und das Manufacturing Execution System zur Produktionsplanung. Außerdem wurden noch ein neu erstelltes Customer-Portal, das den Produktionsstatus eines Auftrages anzeigt und ein, ebenfalls neu erstelltes, Maintenance-Portal, mit dem Wartungsaufgaben erfüllt werden können, eingebunden.
Als Infrastruktur des Systems wurde ein ESB verwendet. Als Schnittstellen von dem ESB zu den zu integrierten Systemen wurden, wie in dieser Arbeit auch, Services verwendet.
Auf Ruthardts werden ESB keine Messages verschickt, sondern Events. Diese Events werden zum einen Teil maschinell und zum anderen Teil vom Benutzer ausgelöst. In dieser Arbeit

hingegen wird ausschließlich auf die Eingaben des Benutzers reagiert.
Ein weiterer Unterschied ist die grundlegende Kommunikation. In Ruthardts Arbeit wurde eine synchrone Architektur gewählt, während in dieser Arbeit der komplette Ablauf asynchron ist. Der Nachteil von Synchronität ist, dass es zum Fehler oder Stillstand kommen kann, wenn ein System aufgerufen werden soll, welches im Moment nicht verfügbar ist. Aus der Asynchronität und der Verwendung von Warteschlangen ergibt sich der Vorteil, dass ein kurzfristiger Systemausfall ohne Folgen bleibt, da die Nachrichten auf den Warteschlangen gespeichert werden.

4.4.2. Champagne

Champagne ist eine Plattform, die ebenfalls verschiedene Systeme integriert um zwischen diesen und dem DPE Daten auszutauschen. Die Systeme lauten:

- Fabrikplanungstisch
- Logistikprüfstand
- Montagekonfigurator

Allerdings wurde die Kommunikation zwischen Champagne und den restlichen Systemen anders gelöst als in dieser Arbeit. In der Champagne Lösung wird die Champagne Plattform direkt vom DPE aufgerufen und je nach Anwendungsfall verbindet sich Champagne direkt mit dem entsprechenden System [MSU06]. Dabei werden keine Nachrichten verschickt, sondern es werden direkt Methoden fremder Systeme aufgerufen. Beispielsweise findet das Laden eines Projektes in den Planungstisch wie folgt statt:

- Der DPE ruft Champagne auf und übergibt das Projekt, das geladen werden soll.
- Champagne verbindet sich mit der Accessdatenbank des Planungstisches.
- Champagne schreibt das Projekt, das ihm vom DPE übergeben wurde direkt in die Tabellen der Accessdatenbank des Planungstisches.

Dadurch ist das System nur schwer wartbar. So muss jedes mal die Champagne Plattform geändert werden, wenn sich die Datenbank nur geringfügig ändert oder das System auf einen neuen Rechner verlagert wird. Dadurch hat der Planungstisch eine feste Bindung an Champagne. In der Architektur dieser Arbeit gibt es keine feste Bindung zwischen verschieden Systemen.
In dieser Arbeit lassen sich die Datenbanken hinter den Services frei austauschen, ohne dass die Kommunikation zwischen den verschiedenen Services beeinträchtigt wird. Ein weiterer Vorteil der Architektur dieser Arbeit sind die frei konfigurierbaren Routingtabellen, diese lassen sich ohne Programmieraufwand ändern und somit auch die Zieladressen der verschiedenen Services neu festlegen. Bei Champagne hingegen müsste eine solche Änderung programmiertechnisch gelöst werden.

5. Zusammenfassung und Ausblick

In diesem Kapitel wird die Arbeit nochmal zusammen gefasst. Anschließend werden mögliche Erweiterungen und Verbesserungen des Systems diskutiert. Die verwendete Architektur birgt noch viel Potential, unter anderem die Möglichkeit das System zu erweitern.

5.1. Zusammenfassung

Diese Arbeit befasst sich mit der Integration von proprietären Systemen. Ausgangssituation dafür war die Lernfabrik des IFFs mit ihren Systemen der Planungsunterstützung.
In der digitalen Fabrik des IFF waren bereits mehrere Systeme miteinander verbunden. Unter anderem der DPE, der alle Planungsdaten zentral verwaltet, der Planungstisch, der in der Layoutplanung eingesetzt wird und der Logistikprüfstand, der die Ergebnisse der Layoutplanung simuliert. Diese Systeme waren alle durch Punkt-zu-Punkt Verbindungen miteinander verbunden und verwenden unterschiedliche Datenformate, was Wartungsarbeiten am Gesamtsystem erschwert und das Austauschen der Systeme nur durch einen hohen Aufwand möglich macht.
Aufgrund der mangelnden Flexibilität dieses Intergrationsansatzes sollte eine neue Architektur entworfen werden, bei dieser der Austausch von Systemen leichter zu realisieren ist. Dabei sollte es möglich sein die Planungsdaten zwischen DPE und Planungstisch auszutauschen.
Die neue Architektur wurde nach dem SOA Prinzip aufgebaut. Dieser Ansatz stellt sicher, dass die verschiedenen Systeme lose gekoppelt sind, da als Schnittstelle der Systeme Services eingesetzt werden, bei denen darauf geachtet wird, dass Serviceendpunkt und Implementierung der Services strickt voneinander getrennt sind. Um die lose Kopplung der Systeme weiter zu unterstützen wurden Message Queues eingesetzt, welche die Nachrichten, die an den Service geschickt werden, entgegennehmen.
Als Infrastruktur der Services wurde ein ESB eingesetzt, über den Nachrichten, basierend auf einem gemeinsamen Nachrichtenmodell, verschickt werden. Das gemeinsame Nachrichtenmodell ermöglicht es alle Services mit dem gleichen Interface zu erstellen. Das Routen übernimmt ebenfalls der ESB, der anhand der Nachrichtenstruktur die Zieladressen der einzelnen Nachrichten bestimmt.
Um den Prozess der Layoutplanung geeignet abbilden zu können wurde ein Workflow modelliert, der aus dem Portal heraus gestartet werden kann.
Aus dem Erreichen der Ziele und der Vorteile der gewählten Technologien, im Gegensatz zu der bisher verwendeten, lässt sich erkennen, dass der SOA Ansatz die Integration der Systeme verbessert hat. Durch die lose Kopplung ist das Gesamtsystem jetzt flexibler, die

integrierten Werkzeuge lassen sich schneller austauschen und die Architektur leichter um neue Systeme erweitern. Dadurch lässt sich die Planung auf eventuelle Kundenwünsche oder geänderte Anforderungen schneller anpassen.
Außerdem wurde der Wartungsaufwand durch die Kommunikation über ein gemeinsames Nachrichtenmodell und des Prinzips der losen Kopplung, deutlich reduziert.
Ziel dieser Arbeit war es ausgewählte Systeme der Planung über Services miteinander zu verbinden und an einen ESB anzuschließen. Dies wurde durch die Integration von DPE und Planungstisch erreicht.
Unternehmen stehen ebenfalls vor der Aufgabe die Daten ihrer planungsunterstützenden Werkzeuge in geeigneter Weise auszutauschen. Deshalb lässt sich der Lösungsansatz dieser Arbeit auch auf Unternehmen übertragen, die die Steigerung der Effizienz ihrer digitalen Fabriken und somit eine verbesserte Planungsunterstützung, zum Ziel haben.

5.2. Ausblick

Wie bereits erwähnt, gibt es in der Planungsphase Systeme, die problemlos in die Architektur mit aufgenommen werden können.
Es wäre denkbar ein Werkzeug, das die erstellten Layoutplanungen anschließen simuliert, in das System zu integrieren.
Ein mögliches Werkzeug für die Simulation könnte der Logistikprüfstand sein, der effizient die Gestaltung sowie die Auswahl und Parametrierung der erforderlichen Auftragsmanagementmethoden unterstützt [KLWW03].
Dabei kann das System nicht nur in die Architektur integriert werden, es wäre auch möglich innerhalb des Portals einen neuen Workflow zu starten, welcher Systeme, die erforderlich sind um eine Simulation auszuführen, integriert. Die zusätzlich benötigten Daten können in das erweiterbare Nachrichtenmodell eingebunden werden. Die Simulationsphase könnte in den Planungs Workflow aufgenommen und modelliert werden.

5.2.1. Integration neuer Systeme

Bei der Integration von neuen Systemen müssen folgende Punkte beachtet werden:

Sammlung von Daten

Zuerst muss das System genau analysiert werden, das heißt, es muss überprüft werden welche Daten das System benötigt. Anschließend müssen diese Daten mit dem gemeinsamen Nachrichtenmodell verglichen werden. Reichen die Daten, die bereits in den Nachrichten übertragen werden, aus oder muss das Schema erweitert werden. Falls noch zusätzliche Informationen benötigt werden, muss das Nachrichtenmodell um diese erweitert werden.
Des weiteren muss festgestellt werden, ob eventuell ein neuer MessageType eingeführt werden muss. Falls dies der Fall ist, so muss die Klasse *BaseService* angepasst und eventuell um Methoden erweitert werden.

Erstellen eines neuen Services

Wenn das Schema angepasst wurde, dann muss ein neuer Service erstellt werden, der als Schnittstelle des neuen Systems fungiert. Außerdem muss eine neue Message Queue erstellt werden, an welche die Nachrichten für diesen Service geschickt werden.
Dabei muss darauf geachtet werden, dass das gemeinsame Interface benutzt wird, damit beim Nachrichtenaustausch keine Probleme auftreten.
Anschließend müssen die benötigten Methoden aus der Klasse *BaseService* überschrieben werden. Dabei ist zu prüfen wie Service und die zu integrierenden System kommunizieren sollen. Es muss auf jeden Fall eine asynchrone Kommunikation gewählt werden, da die komplette Architektur des Gesamtsystems auf diese Weise funktioniert.

Anpassung des Gesamtsystems

Lediglich an zwei Stellen müssen Änderungen vorgenommen werden. Zum einen ist es der Router und zum anderen der DPE Service.
Die Routingtabellen des Routers müssen um das neue System erweitert werden. Dabei wird der neue Service mit seinem Endpunkt eingetragen. Außerdem muss die Logik der XPATH-Ausdrücke verifiziert und gegebenenfalls angepasst werden.
Änderungen am DPE sind nur notwendig, wenn bei der Sammlung der Daten festgestellt wurde, dass noch mehr Informationen, als bisher im Nachrichtenmodell verwendet, benötigt werden. Dann muss die Funktionalität des DPE Service um das Auslesen und das Zurückschreiben dieser zusätzlichen Informationen erweitert werden.

5.2.2. Erweiterung des Portals um neuen Workflow

Das Portal wurde extra so konzipiert, dass es einfach ist neue Workflows zu integrieren.
Es wurde extra die Tabelle *Modules* erstellt, in der alle Workflows, die gestartet werden können, gespeichert sind. Hier muss der Name des neuen Workflows eingetragen werden.
Natürlich müssen zusätzlich noch weitere Webseiten erstellt werden, auf dem der Fortschritt des Workflows zu sehen ist. Beim Auswählen des neuen Workflows ist darauf zu achten, dass die neuen Webseiten entsprechend angezeigt werden. Das gleiche muss beachtet werden, wenn der Portal Service eine neue Nachricht erhält und in die Tabelle *Updates* schreibt. Beim Lesen des Portals von dieser Tabelle muss ebenfalls wieder sichergestellt sein, dass die neuen Webseiten angezeigt werden und der Status dort entsprechend des Fortschritts des Workflows angepasst wird.

5.2.3. Service in Planungs Workflow integrieren

Es ist ebenfalls möglich einen neuen Service in den aktuellen Planungs Workflow zu integrieren. Da alle Services des Systems, inklusive dem neu hinzugekommenen, das selbe Schnittstellenmodell benutzen, ist es problemlos möglich den aktuellen Workflow um einen

neuen Service zu erweitern.
Da es sich um einen dynamischen Aufruf der Services handelt und die Adresse der Services von dem Router ermittelt wird, ist es möglich den neuen Service in den Workflow zu integrieren.

5.3. Visionen

In diesem Abschnitt werden Verbesserungsvorschläge und eventuelle Neuerungen im Zusammenhang mit dem jetzigen Gesamtsystem diskutiert.

5.3.1. Einführung eins Service Managements

In dieser Arbeit wurde noch vorgesehen ein Service Management zu erstellen, was aber aufgrund von Zeitmangel nicht möglich war. Doch für weitere Arbeiten wäre dies ein mögliches Themengebiet.
Ein Service Management würde zur Übersicht aller verwendeten Systeme beitragen. In diesem sollen alle Systeme aufgelistet werden und eventuell die Abhängigkeiten graphisch dargestellt werden können.

5.3.2. Anbindung an des Manufacturing Service Bus

Der Manufacturing Service Bus, den Frank Ruthardt in seiner Diplomarbeit *Event-driven Production Rescheduling* [Rut10] erstellt hat, ist ein Enterprise Service Bus, der die Systeme der Produktionsplanung miteinander verbindet. Zu diesen Systemen zählen unter anderem der Leitrechner für die Fabriksteuerung und das Manufacturing Execution System zur Produktionsplanung. Werden die zwei Service Busse miteinander gekoppelt, so wäre es möglich die Daten zwischen Produktionsplanung und Layoutplanung auszutauschen.

A. Listings

Listing A.1: Nachrichtenmodell-Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://tempuri.org/ESB" xmlns:xs="http://www.w3.org/2001/XMLSchema"
	targetNamespace="http://tempuri.org/ESB" elementFormDefault="qualified"
	attributeFormDefault="unqualified">
	<xs:element name="ProjectMessage">
		<xs:annotation>
			<xs:documentation>Beschreibt ein Objekt</xs:documentation>
		</xs:annotation>
		<xs:complexType>
			<xs:sequence>
				<xs:element name="Projects" type="tns:Projects" minOccurs="1"
				    maxOccurs="unbounded"/>
				<xs:element name="CommonInfos" type="tns:CommonInfos"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:complexType name="Projects">
		<xs:sequence>
			<xs:element name="ProjectID" type="xs:int"/>
			<xs:element name="Projectname" type="xs:string"/>
			<xs:element name="Ressources" type="tns:Ressources"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="Ressources">
		<xs:annotation>
			<xs:documentation>Beinhaltet die Resourcen</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:element name="ModuleType" type="tns:ModuleType" minOccurs="0"
			    maxOccurs="unbounded"
			/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="ModuleType">
		<xs:annotation>
			<xs:documentation>Beinhaltet die verschiedenen Typen</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:element name="ModuleID" type="xs:int"/>
			<xs:element name="Modulename" type="xs:string"/>
			<xs:element name="Type" type="xs:int"/>
			<xs:element name="FileName" type="xs:string"/>
			<xs:element name="BitmapName" type="xs:string"/>
			<xs:element name="Module" type="tns:Module" maxOccurs="unbounded"/>
```

```
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Module">
        <xs:annotation>
            <xs:documentation>Instanzen der Modultypen</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="ObjectID" type="xs:int"/>
            <xs:element name="Objectname" type="xs:string"/>
            <xs:element name="Position" type="tns:Position"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Position">
        <xs:annotation>
            <xs:documentation>Beschreibt die Position der
                Instanzen</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="Position_X" type="xs:double"/>
            <xs:element name="Position_Y" type="xs:double"/>
            <xs:element name="Position_Z" type="xs:double"/>
            <xs:element name="Rotation_X" type="xs:double"/>
            <xs:element name="Rotation_Y" type="xs:double"/>
            <xs:element name="Rotation_Z" type="xs:double"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CommonInfos">
        <xs:annotation>
            <xs:documentation>Beinhaltet allgemeine Informationen</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="MessageType" type="xs:int"/>
            <xs:element name="MessageIdregistered" type="xs:boolean"/>
            <xs:element name="MessageId" type="xs:string"/>
            <xs:element name="MessageFlowIdregistered" type="xs:boolean"/>
            <xs:element name="MessageFlowId" type="xs:string"/>
            <xs:element name="RoutingInfos" type="tns:RoutingInfos"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="RoutingTypeDestination">
        <xs:sequence>
            <xs:element name="SystemID" type="xs:string"/>
            <xs:element name="SystemUri" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="RoutingInfos">
        <xs:annotation>
            <xs:documentation>Beschreibt Quelle und Ziel</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="OriginSystem" type="xs:string" minOccurs="0"
                maxOccurs="1"/>
            <xs:element name="DestinationSystem" type="tns:RoutingTypeDestination"
                minOccurs="0"
                   maxOccurs="unbounded"/>
```

```
            </xs:sequence>
        </xs:complexType>
</xs:schema>
```

Listing A.2: Beispiel einer Nachricht

```
<?xml version="1.0" encoding="utf-8"?>
<ProjectMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <CommonInfos xmlns="http://tempuri.org/ESB">
    <RoutingInfos>
      <OriginSystem>DPE</OriginSystem>
      <DestinationSystem>
        <SystemID>WP</SystemID>
        <SystemURI>http://localhost:9007/PortalService/</SystemURI>
      </DestinationSystem>
    </RoutingInfos>
    <MessageType>0</MessageType>
    <MessageIdregistered>true</MessageIdregistered>
    <MessageId>WP_14.07.2011 11:47:36</MessageId>
    <MessageFlowIdregistered>true</MessageFlowIdregistered>
    <MessageFlowId>WP_14.07.2011 11:47:36_2</MessageFlowId>
  </CommonInfos>
  <Projects xmlns="http://tempuri.org/ESB">
    <Resources>
      <ModuleTypes>
        <ModuleID>1</ModuleID>
        <Modulename>modname</Modulename>
        <Type>1</Type>
        <FileName>test.cad</FileName>
        <BitmapName>test.bmp</BitmapName>
        <Module>
          <ObjectID>1</ObjectID>
          <Objectname>objectname</Objectname>
          <Position>
            <Position_X>300</Position_X>
            <Position_Y>80</Position_Y>
            <Position_Z>0</Position_Z>
            <Rotation_X>0</Rotation_X>
            <Rotation_Y>0</Rotation_Y>
            <Rotation_Z>0</Rotation_Z>
          </Position>
        </Module>
      </ModuleTypes>
    </Resources>
    <ProjectID>1</ProjectID>
    <Projectname>Test Projektt</Projectname>
  </Projects>
</ProjectMessage>
```

Listing A.3: Abstrakte Service-Definition

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions targetNamespace="http://tempuri.org/ESB"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
    xmlns:tns="http://tempuri.org/ESB"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:wsap="http://schemas.xmlsoap.org/ws/2004/08/addressing/policy"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:msc="http://schemas.microsoft.com/ws/2005/12/wsdl/contract"
    xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:wsa10="http://www.w3.org/2005/08/addressing"
    xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex">
    <wsdl:types>
        <xsd:schema targetNamespace="http://tempuri.org/ESB/Imports">
            <xsd:import schemaLocation="Transport_Schema.xsd"
                namespace="http://tempuri.org/ESB"/>
        </xsd:schema>
    </wsdl:types>
            <wsdl:message name="MsmqMessage">
        <wsdl:part name="ProjectMessage" element="tns:ProjectMessage"/>
    </wsdl:message>
    <wsdl:portType name="IService">
        <wsdl:operation name="ProcessMessage">
            <wsdl:input wsaw:Action="ProcessMessage" name="MsmqMessage"
                message="tns:ProjectMessage"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

Literaturverzeichnis

[Auto3] Automatisierung, Fraunhofer Institut P.: *I-PlAnt Dokumentation*. Fraunhofer Institut Produktionstechnik und Automatisierung, 2003 (Zitiert auf Seite 47)

[BGW11] Bracht, Uwe ; Geckler, Dieter ; Wenzel, Sigrid: *Digitale Fabrik Methoden und Praxisbeispiele*. Springer, 2011 (Zitiert auf Seite 9)

[BHR07] Buchholz, Andreas ; Hohloch, Rüdiger ; Rathgeber, Tim: *Vergleich von Open Source ESBs*. 2007 (Zitiert auf den Seiten 18 und 19)

[Bib08] Biberger, Ulrich: *Der Enterprise Service Bus in Theorie und Praxis*. Grin Verlag, 2008 (Zitiert auf den Seiten 17 und 27)

[BS05] Bickel, Boris ; Schuster, Marcel: *Trends, Methoden und Grundsätze moderner Fabrik- und Produktionsplanung*. Grin Verlag, 2005 (Zitiert auf Seite 47)

[Bur07] Burbiel, Herber: *SOA & Webservices in der Praxis*. Franzis Verlag GmbH, 2007 (Zitiert auf den Seiten 15, 21 und 22)

[CG] Carlos, Jorge ; Gómez, Marx: *Serviceorientierte Architektur*. http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Systementwicklung/Softwarearchitektur/Architekturparadigmen/Serviceorientierte-Architektur (Zitiert auf Seite 16)

[FZ09] Finger, Patrick ; Zeppenfeld, Klaus: *SOA und Webservices*. Springer, 2009 (Zitiert auf Seite 15)

[gla] http://wiki.open-esb.java.net/Wiki.jsp?page=GlassFishESBDocs (Zitiert auf Seite 19)

[Gru09] Grundig, Claus-Gerold: *Fabrikplanung: Planungssystematik - Methoden - Anwendungen*. Carl Hanser Verlag, 2009 (Zitiert auf den Seiten 9 und 13)

[jav] http://java.com/de/about/ (Zitiert auf Seite 26)

[Jos09] Josuttis, Nicolai: *SOA in der Praxis*. dpunkt.verlag, 2009 (Zitiert auf den Seiten 15, 16 und 17)

[Kai04] Kaib, Michael: *Enterprise Application Integration: Grundlagen, Integrationsprodukte, Anwendungsbeispiele*. Deutscher Universitäts-verlag, 2004 (Zitiert auf Seite 17)

[Kan08] Kansy, Thorsten: *Datenbankprogrammierung mit .NET 3.5*. Carl Hanser Verlag, 2008 (Zitiert auf Seite 28)

[KB07] Kuhrmann, Marco ; Beneken, Gerd: *Windows Communication Foundation: Konzepte-Programmierung-Migration*. Spektrum Akademischer Verlag, 2007 (Zitiert auf Seite 26)

[Küh06] Kühn, Wolfgang: *Digitale Fabrik : Fabriksimulation für Produktionsplaner*. Hanser, 2006 (Zitiert auf den Seiten 9 und 14)

[KH09] Kotz, Jürgen ; Hölzl, Stephani: *WCF - Windows Communication Foundation : verteilte Anwendungsentwicklung mit der Microsoft-Kommunikationsplattform*. Addison Wesley in Pearson Education Deutschland, 2009 (Zitiert auf den Seiten 22 und 24)

[KLWW03] Kapp, R. ; Löffler, B. ; Wiendahl, H.-H. ; Westkämper, E.: Der Logistik-Prüfstand Skalierbare Logistiksimulation von der Lieferkette bis zum Arbeitsgang. In: *wt Werkstattstechnik online* (2003) (Zitiert auf Seite 82)

[Kon09] Konopasek, Klemens: *SQL Server 2008 - Der schnelle Einstieg: Abfragen, Transact-SQL, Entwicklung und Verwaltung*. Addison Wesley Verlag, 2009 (Zitiert auf Seite 29)

[Kon10] Konopasek, Klemens: *SQL Server 2008 R2 - Der schnelle Einstieg*. Addison Wesley Verlag, 2010 (Zitiert auf Seite 29)

[KT05] Knecht-Thurmann, Stephanie: *Small Business Server 2003*. Addison Wesley Verlag, 2005 (Zitiert auf Seite 19)

[ler] http://www.lernfabrik-aie.de/schulungszentrum/ (Zitiert auf Seite 9)

[LH09] Leibing, Stefan ; Held, Bernd: *Access VBA*. Pearson Studium, 2009 (Zitiert auf Seite 25)

[Loo01] Loos, Peter: *Go to COM*. Addison Wesley Verlag, 2001 (Zitiert auf Seite 25)

[LS08] Liebhart, Daniel ; Schmutz, Guido: *Integration Architecture Blueprint: Leitfaden zur Konstruktion von Integrationslösungen*. Hanser, 2008 (Zitiert auf Seite 27)

[Mat08] Mathas, Christoph: *SOA intern*. Carl Hanser Verlag, 2008 (Zitiert auf den Seiten 15 und 27)

[Mel08] Melzer, Ingo: *Service-orientierte Architekturen mit Web Services*. Spektrum Akademischer Verlag, 2008 (Zitiert auf den Seiten 15, 16 und 17)

[MKG09] Müller, Thomas ; Käser, Hans ; Gübli, Rolf: *Technische Informatik I: Grundlagen der Informatik und Assemblerprogrammierung*. VDF Verlag, 2009 (Zitiert auf Seite 18)

[msd] http://msdn.microsoft.com (Zitiert auf Seite 70)

[MSH03] Middendorf, Stefan ; Singer, Reiner ; Heid, Jörn: *Java - Programmierhandbuch und Referenz*. dpunkt.verlag, 2003 (Zitiert auf Seite 26)

[MSU06] Müller, Egon ; Spanner-Ulmer, Birgit: *Vernetzt planen und produzieren VPP 2006*. Technische Universität Chemnitz, 2006 (Zitiert auf Seite 79)

[PP04] Pomberger, Gustav ; Pree, Wolfgang: *Software Engineering*. Carl Hanser Verlag, 2004 (Zitiert auf Seite 22)

[rem] http://msdn.microsoft.com/de-de/netframework/default.aspx (Zitiert auf Seite 20)

[Rut10] Ruthardt, Frank: *Event-driven Production Rescheduling*, Universität Stuttgart, Diplomarbeit, 2010 (Zitiert auf den Seiten 78 und 84)

[Sch10] Schwichtenberg, Holger: *Windows Scripting*. Addison Wesley in Pearson Education Deutschland, 2010 (Zitiert auf Seite 25)

[Sta04] Stamer, Jan: *Web Services mit microsoft.net*. Grin Verlag, 2004 (Zitiert auf Seite 20)

[SW04] Schenk, Michael ; Wirth, Siegfried: *Fabrikplanung und Fabrikbetrieb: Methoden für die wandlungsfähige und vernetzte Fabrik*. Springer, 2004 (Zitiert auf Seite 10)

[Thu10] Thulasidas, Manoj: *Principles of Quantitative Development*. Wiley Finance, 2010 (Zitiert auf Seite 25)

[WZ09] Westkämper, Engelbert ; Zahn, Erich: *Wandlungsfähige Produktionsunternehmen: Das Stuttgarter Unternehmensmodell*. Springer, 2009 (Zitiert auf Seite 47)

Alle URLs wurden zuletzt am 17.10.2011 geprüft.

Abbildungsverzeichnis

Verzeichnis der Listings

Erklärung

Hiermit versichere ich, diese Arbeit
selbstständig verfasst und nur die angegebenen
Quellen benutzt zu haben.

(Fabian Laux)