HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr
Eric Gebhardt

Entwicklung eines Wrappers für die Integration von Drittprogrammen in das firmeneigene Mitarbeiter- und Projekt-Buchungssystem

2014

Fakultät **Elektro- und Informationstechnik**

DIPLOMARBEIT

Entwicklung eines Wrappers für die Integration von Drittprogrammen in das firmeneigene Mitarbeiter- und Projekt-Buchungssystem

Autor:
Eric Gebhardt

Studiengang:
Informationstechnik

Seminargruppe:
KI10w1-F

Erstprüfer:
Prof. Dr.-Ing. Rolf Hiersemann

Zweitprüfer:
Dipl.-Ing. Marco Grafe

Mittweida, 01.08. 2014

Bibliografische Angaben

Gebhardt, Eric: Entwicklung eines Wrappers für die Integration von Drittprogrammen in das firmeneigene Mitarbeiter- und Projekt-Buchungssystem, 89 Seiten, 30 Abbildungen, 9 Tabellen, Hochschule Mittweida, University of Applied Sciences, Fakultät Elektro- und Informationstechnik

Diplomarbeit, 2014

Satz: LaTeX

Referat

Die vorliegende Arbeit befasst sich mit der Entwicklung eines Wrappers für die Integration von Drittprogrammen in das firmeneigene Mitarbeiter- und Projekt-Buchungssystem. Dazu wird die bereits bestehende Datenbasis gesichtet und erweitert. Auf Grundlage einer Anforderungsanalyse werden die entsprechenden Usecases ermittelt und ein Nachrichtenkonzept herausgearbeitet. Für die Datenübertragung werden geeignete Protokolle sowie eine Auszeichnungssprache ausgewählt. Bei der weiteren Softwareentwicklung werden verschiedene Methoden der UML angewendet. Nach der Implementierung wurde das neue System integriert und iterativ eingeführt. Abschließend wird eine Beurteilung des entstandenen Systems durchgeführt und ein Ausblick über zukünftige Funktionalitäten gegeben.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Abkürzungsverzeichnis

AAA	engl. Arrange, Act and Assert - Arrangieren, Ausführen und Annehmen
BSD	Bestandsdaten
BWD	Bewegungsdaten
CA	engl. Certification Authority - Zertifizierungsstelle
CAD	engl. Computer-aided design - rechnerunterstütztes Konstruieren
COBRA	engl. Common Object Request Broker Architecture - Allgemeine Architektur für Vermittler von Objekt-Nachrichten
DBMS	Datenbankmanagementsystem
DDD	engl. Domain Driven Development - Domän-gesteuerte Entwicklung
DDL	engl. Data Definition Language - Datenbeschreibungssprache
DML	engl. Data Manipulation Language - Datenverarbeitungssprache
DoD	engl. Department of Defense - Verteidigungsministerium
DRL	engl. Data Retrievel Language - Datenabfragesprache
DSL	engl. Data Security Language - Datensicherheitssprache
ERP	engl. Enterprise-Resource-Planing - Unternehmen-Ressourcen-Planung
FIFO	engl. first in / first out - zuerst herein / zuerst hinaus
FTP	engl. File Transfer Protocol - Dateiübertragungsverfahren
GUI	engl. Graphical User Interface - Graphische Benutzeroberfläche
HTML	engl. Hypertext Markup Language - Hypertext Auszeichnungssprache
HTTP	engl. Hypertext Transfer Protocol - Hypertext-Übertragungsprotokoll i. d. R. eingesetzt um Webseiten zu übertragen
HTTPS	engl. Hypertext Transfer Protocol Secure - ‘sicheres’ Hypertext-Übertragungsprotokoll i. d. R. eingesetzt um sensible Webseiten zu übertragen
ID	engl. Identifier - Identifikator, Kennung
IDE	engl. Integrated Development Environment - integrierte Entwicklungsumgebung, z. B. ‘Microsoft Visual Studio’
IMAP	engl. Internet Message Access Protocol - Netzwerkprotokoll, das ein Netzwerkdateisystem für E-Mails bereitstellt
IP	engl. Internet Protocol - Internetprotokoll
ISO	engl. International Organization for Standardization - Internationale Organisation für Normung
ITU	engl. International Telecommunication Union - Internationale Fernmeldeunion
L2	engl. Layer 2 - Schicht 2 des Netzwerkprotokolls

LDAP ..... engl. Lightweight Directory Access Protocol - leichtgewichtiges Verzeichniszugriffsprotokoll

MA ..... Mitarbeiter

NCP ..... engl. NetWare Core Protocol - Netzwerkprotokoll des Netzbetriebssystem NetWare

OLE ..... engl. Object Linking Embedding - Objekt-Verknüpfung und -Einbettung

OPC UA ..... engl. OLE for Process Control Unified Architecture - Protokoll der Automatisierungstechnik

OSI ..... engl. Open Systems Interconnection - Standard eines Referenzmodells für Netzwerkprotokolle veröffentlicht von ITU und ISO

POP3 ..... engl. Post Office Protocol Version 3 - E-Mail-Transportprotokoll zum Abholen von Emails von einem Server

RFC ..... engl. Requests For Comments - standardähnliches Dokument herausgegeben von der Internet Society

SD ..... Stammdaten

SDK ..... engl. Software Development Kit - Sammlung von Werkzeugen zur Softwareentwicklung

SGML ..... engl. Standard Generalized Markup Language - normierte, verallgemeinerte Auszeichnungssprache

SMTP ..... engl. Simple Mail Transfer Protocol - einfaches E-Mail-Transportprotokoll

SQL ..... engl. Structured Query Language - strukturierte Anfragesprache für Datenbanken

SSL ..... engl. Secure Socket Layer - sichere Anschlussschicht

TCP ..... engl. Transmission Control Protocol - Übertragungssteuerungsprotokoll: ein verbindungsorientiertes, paketvermitteltes Netzwerkprotokoll

Telnet ..... engl. Telecommunicaton Network - altes Netzwerkprotokoll z. B. für den Fernzugriff

TLS ..... engl. Transport Layer Security - Transportschichtsicherheit

UDP ..... engl. User Datagramm Protocol - Benutzerdatensegmentprotokoll: ein verbindungsloses Netzwerkprotokoll

UDS ..... engl. Unified Diagnostic Services - Kommunikationsprotokoll der Automobilelektronik

UML ..... engl. Unified Modeling Language - Vereinheitlichte Modellierungssprache

VoIP ..... engl. Voice over Internet Protocol - Internet-Telefonie

XML ..... engl. Extensible Markup Language - erweiterbare Auszeichnungssprache

1 Einleitung

1.1 Motivation

Ein Unternehmer ist, „wer [...] unternehmerische Entscheidungen trifft“[1] um seine Ziele, z. B. Schaffung eines wirtschaftlichen Mehrwertes, zu erreichen. Entscheidungen müssen u. a. über die zentralen Ressourcen wie Rohmaterialien, Energie sowie die menschliche und maschinelle Arbeitskraft getroffen werden. Durch den gezielten Einsatz bzw. die strategische Kombination dieser Ressourcen kann ein Mehrwert entstehen. Für die Erreichung dieses Zieles ist eine strategische Planung, die Erfassung und Überwachung des Produktionsprozesses sowie dessen Analyse im Nachgang unverzichtbar. Heutzutage unterstützen den Unternehmer bzw. die Buchhaltungsabteilung (Controlling) des Unternehmens sogenannte *ERP*-Systeme bei der Bewerkstelligung solcher komplexen Aufgaben. Diese Systeme dienen der Planung, Erfassung, Verwaltung und Analyse von einzelnen Prozessen bis hin zur Abbildung komplexer Strukturen. Die Datenhaltung geschieht i. d. R. in einer Datenbank auf einem Server-System. Die Bedienung durch den Nutzer erfolgt hingegen von mehreren Clients, welche mit dem Server kommunizieren.

Die Firma FHR setzt für die Ressourcenplanung zwei verschiedene *ERP*-Systeme ein. Der Fokus der Diplomarbeit liegt auf dem System der Firma DIGITAL-Zeit, welches die Ressource „Mensch“ verwaltet. Das System führt die Mitarbeiterkonten, wie Urlaubs- und Überstundenkonto, sowie die Projektbuchungsdaten. Es besitzt die eingangs erwähnte Server-Client-Architektur. Der Server wird durch eine Datenbank namens „Avero“ sowie einem gleichnamigen Hintergrunddienst realisiert. Mittels der Client-Software „PC-Buchen“ können die Mitarbeiter Projektbuchungen tätigen sowie ihre Mitarbeiterkonten einsehen. Weiterhin gibt es ortsgebundene Terminals, an welchen mit Hilfe mitarbeiterbezogener Chipkarten ebenfalls Buchung vorgenommen werden können.

Während der Nutzungsdauer von mittlerweile 10 Jahren haben sich die folgende Nachteile der Client-Software herauskristallisiert.

- keine Anpassungsfähigkeit an firmenspezifische Informationen
- eingeschränkte bzw. nicht zeitgemäße Bedienergonomie mit geringem Bedienkomfort
- eingeschränkte Exportfunktionalität
- keine Lauffähigkeit unter dem Betriebssystem „Windows 7“

Da die Nachteile immer mehr überwiegen, soll eine neue Client-Software (Drittanwen-

[1] Wikipedia 2014, „Unternehmer“

dung) zum Einsatz kommen. Die ursprüngliche Datenbank soll allerdings erhalten bleiben und weiter genutzt werden, da weitere Systeme in diese Datenbank Informationen speisen bzw. abrufen. Der Datenaustausch zwischen Drittanwendung und Datenbank soll dabei aus Sicherheitsgründen nicht direkt, sondern über eine Vermittler-Software erfolgen, einem sogenannten „Wrapper“. Der Wrapper soll Befehle, z. B. Buchungsanfragen, der Drittanwendung in SQL-Kommandos umwandeln und an die Datenbank absetzen. Die Antwort des Datenbank-Servers soll anschließend ausgewertet, für die Drittanwendung aufbereitet und an diese versendet werden.

Die vorliegende Arbeit befasst sich mit dem Entwurf und der Dokumentation der Schnittstelle zwischen Drittanwendung und Wrapper. Ein weiterer Bestandteil ist der Softwareentwurf des Wrappers sowie dessen Realisierung. Dies beinhaltet die Kommunikation mit der Datenbank, wofür eine Analyse der bestehenden Datenbasis im Vorfeld durchzuführen ist.

1.2 Projektplanung

Die Entwicklung einer neuen Software ähnelt in vielen Punkten einem neuen Projekt im Projektmanagement. Die Definition eines Projektes,“... [als] ein zielgerichtetes, einmaliges Vorhaben, das aus einem Satz von abgestimmten, gelenkten Tätigkeiten mit Anfangs- und Endtermin besteht und durchgeführt wird, um unter Berücksichtigung von Zwängen bezüglich Zeit, Ressourcen und Qualität ein Ziel zu erreichen“[2] trifft i. A. auch auf die Softwareentwicklung zu. Es gibt nicht nur Stakeholder (dt. Teilhaber), Anforderungen, Analysen, Entwicklungs- sowie Umsetzungsphasen im Projektmanagement, sondern auch im Falle der hier gearteten Entwicklungsaufgabe. Die Entwicklung einer Software ist somit i. d. R. ein Projekt.

Um ein Projekt erfolgreich durchzuführen und mit der angestrebten Qualität umzusetzen, ist eine entsprechende Planung notwendig. Die Planung beinhaltet zum Einen das Identifizieren der einzelnen Tätigkeiten, die für die Erzeugung des Produktes notwendig sind. Zum anderen werden diese Tätigkeiten miteinander in Beziehung gebracht und chronologisch geordnet. Die sortierte Projektplanung dieser Arbeit, welche auch die Grundlage der praktischen Realisierung war, ist die nachfolgende.

1. Projektplanung erstellen
2. konzeptioneller Entwurf
3. Anforderungsanalyse durchführen
4. Analyse der bestehenden Datenbasis
5. Entwurf der zusätzlichen Datenbasis
6. Persona und Anwendungsfälle definieren/beschreiben

[2] Thor 2003, S.22

7. Systemarchitektur festlegen (Module definieren)
8. Sequenzdiagramme erstellen
9. Klassendiagramme ableiten
10. Implementierung
11. Integration in das bestehende System
12. Inbetriebnahme, Testlauf und Bugfixing

Nach dem die Planung abgeschlossen ist, wird die Anforderungsanalyse durchgeführt. Dabei werden zuerst die Stakeholder ermittelt und in gemeinsamen Workshops ein Pflichtenheft erarbeitet. Dieses mündet in eine Liste aus funktionalen und nichtfunktionalen Bedingungen bzw. Anforderungen. Anschließend wird unter den geschaffenen Gesichtspunkten der bestehende Datenbestand gesichtet und beurteilt. Da zum einen der Funktionsumfang erweitert werden soll und zum anderen durch die Drittanwendung der administrative Aufwand sich ebenfalls vergrößert, ist zu prüfen, wie der bestehende Datenbestand sinnvoll erweitert wird.

Anschließend werden die Persona und Anwendungsfälle definiert bzw. beschrieben. Diese bilden die Entscheidungsgrundlage für die Festlegung der Systemarchitektur. Die dabei eingeführten Module werden mittels der erstellten Sequenzdiagramme visualisiert sowie die Architektur in Klassendiagramme abgeleitet.

Auf Grundlage der theoretischen Vorleitungen findet als nächstes die Implementierung mittels der Entwicklungsumgebung „Microsoft Visual Studio“ statt. Ist diese abgeschlossen wird das entstandene System in das bestehende integriert. Mittels Testläufe und iterativer Rollout wird die Software eingeführt. Das Feedback von Nutzern unterstützt dabei das Bugfixing. Die Arbeit endet in einer Zusammenfassung sowie Bewertung der umgesetzten Anforderungen und erreichten Ziele.

2 Konzept

2.1 Ziele

2.1.1 Istzustand

Dieses Kapitel soll die Vision zur Grundstruktur der angestrebten Lösung zeigen und erläutern. Die Ausgangslage soll dazu in Abbildung 2.1 vergegenwärtigt werden. Zurzeit gibt es das *ERP*-System namens „Avero“ der Firma „DIGITAL-Zeit“. Es umfasst die Komponenten Server-Datenbanksystem, Server-Hintergrunddienst, eine Multi-User-Client-Software namens „PC-Buchen“, einem Administrierungstool „Avero“ sowie mehreren Terminals. Die Terminals dienen den manuellen Buchungen mit mitarbeiterbezogenen Chipkarten und sind autarke, eingebettete Systeme. Sie berühren allerdings das zu entwickelnde System nicht und bleiben daher in der Arbeit unberücksichtigt.

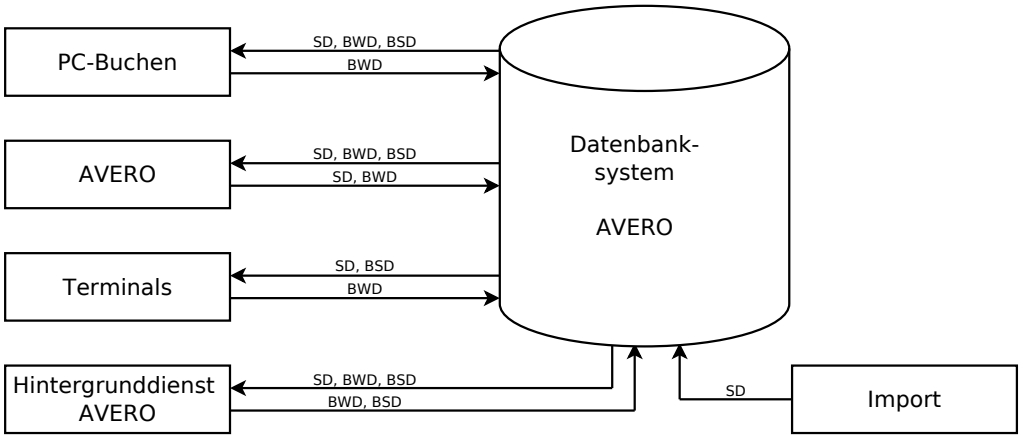

Abbildung 2.1: AVERO-Systemkomponenten sowie deren Austausch von Stammdaten (SD), Bewegungsdaten (BWD) und Bestandsdaten (BSD) mit dem Datenbanksystem

Die Datenbank speichert alle Stamm- (SD), Bestands- (BSD) sowie Bewegungsdaten (BWD) und bildet das Herzstück des bestehenden Systems. Wobei die Stammdaten die Basisinformationen der Ressourcen beinhalten. Die Bestandsdaten kennzeichnen hingegen i. d. R. eine Anzahl, z. B. die aktuellen Urlaubstage. Buchungen oder andere Informationen, welche zur Änderung der Bestandsdaten führt, werden Bewegungsdaten genannt.

Die Datenspeisung erfolgt durch die erwähnten Komponenten Client-Software, Administrierungstool und den Terminals sowie durch Imports vom anderen *ERP*-System. Die eigentliche Datenverarbeitung übernimmt i. d. R. der Hintergrunddienst. Dies ist eine kleine Anwendung mit einer sehr einfach gehaltenen *GUI*. Mit ihrer Hilfe wird die Verarbeitung gestartet oder beendet. Ziel der Applikation ist: angefallene Bewegungsdaten

zu erfassen, zu verarbeiten und die Bestandsdaten dahingehend anzupassen.

2.1.2 Sollzustand

Das entstehende System soll die drei erwähnten Datenkategorien visualisieren und neue Bewegungsdaten erzeugen. Ziel ist es, dass der Hintergrunddienst die neuen Bewegungsdaten ebenfalls verarbeitet und folgerichtig den Bestand anpasst. Dazu ist ein Zugriff auf das Avero-Datenbank-System notwendig. Um diesen Zugriff sicher und stabil zu handhaben, soll er zentral von einer Stelle aus erfolgen. Diese Stelle soll als Vermittler dienen. Er soll dazu Anfragen (Request) von den dezentralen Client-Applikationen (Drittanwendung) empfangen, verarbeiten sowie anschließend jeweils eine Antwort (Response) zurücksenden. Der Vermittler fungiert also als Sammelstelle von Anfragen, welche verarbeitet und beantwortet werden müssen. Es entsteht eine Server-Client-Architektur, siehe Abbildung 2.2.

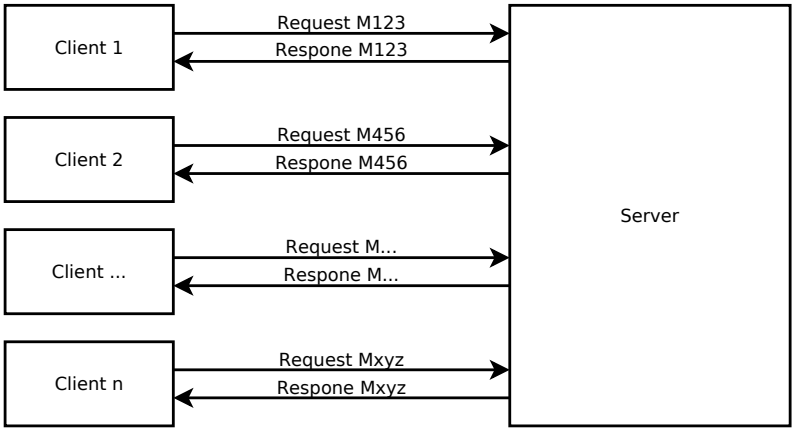

Abbildung 2.2: Klassische Client-Server-Kommunikation mit mehreren Clients

Um die Anfragen zu bearbeiten, bedient sich der Vermittler an Informationen aus der Avero-Datenbank. Er vermittelt sozusagen aufbereitete Informationen. Dies soll allerdings bidirektional geschehen. Es sollen wie eingangs erwähnt neue Bewegungsdaten im Client entstehen und in der Avero-Datenbank abgelegt werden. Programme, die den Charakter wie der Vermittler haben, nennt man in der Informationstechnik „Wrapper“. Sie dienen i. d. R. als Schnittstelle bzw. Adapter zwischen zwei verschiedenen Programmen. Es findet somit keine direkte Kommunikation zwischen den beiden Instanzen Clientapplikation und Datenbank statt, wie Abbildung 2.3 zeigt.

Im Wrapper erfolgen die Verarbeitungsschritte zwischen einer Anfrage und der dazugehörigen Antwort sequentiell nach einer klar definierten Reihenfolge. Diese Abfolge mit den vereinfachten Zwischenschritten sieht wie folgt aus. Die aufgeführten Schritte beinhalten meist mehrere Unterschritte. Diese werden im Kapitel 4.2 näher beleuchtet.

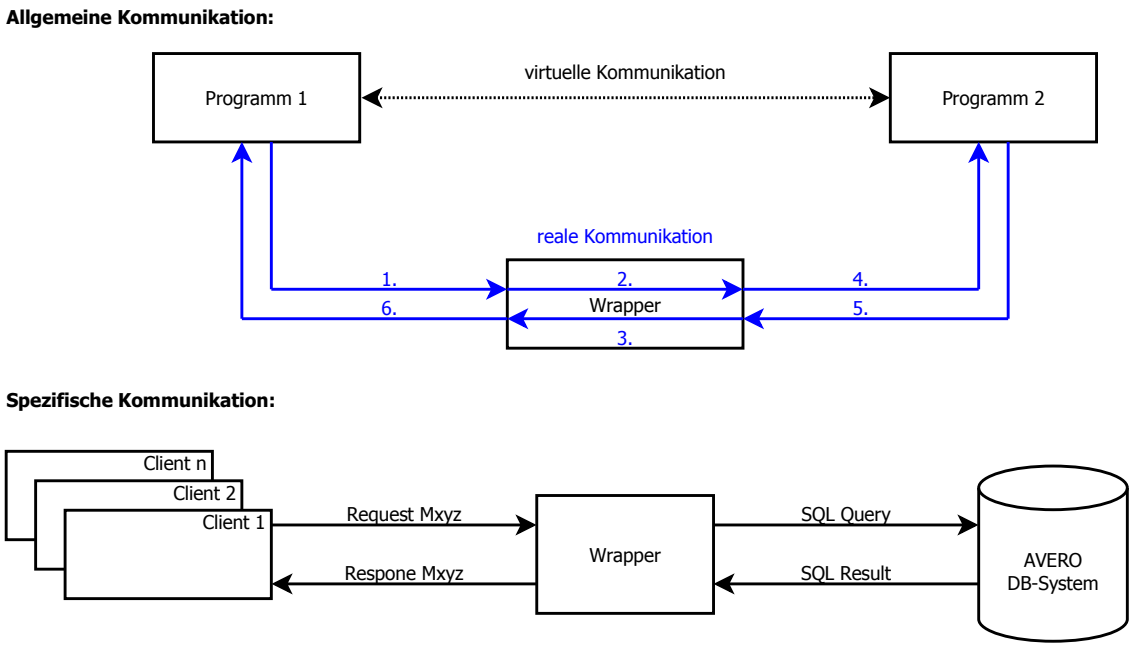

Abbildung 2.3: Allgemeines und spezifisches Kommunikationsprinzip mit einem Wrapper mittels virtuellem und realem Kommunikationsweg

1. Nachricht entgegennehmen
2. Nachricht verifizieren und identifizieren
3. SQL-Befehl an Datenbank absetzen
4. SQL-Reaktion von Datenbank erhalten und auswerten
5. Antwortnachricht bilden
6. Antwortnachricht zurücksenden

Neben dem Server (Wrapper) soll auch eine Clientapplikation entstehen, welche im Firmennetz durch die verschiedenen Mitarbeiter dezentral genutzt werden soll. Die Clientanwendung dient der Visualisierung im Sinne einer grafischen Benutzeroberfläche (*GUI*). Sie zeigt die durch den Wrapper aufbereiteten und übermittelten Informationen aus der Datenbank an. Um diese Daten abzufragen, bildet die Applikation Anfragen (Request) und erhält vom Vermittler verzögert eine entsprechende Antwort (Response). Die entstehende Clientapplikation wird nicht im Umfang dieser Arbeit erstellt. Die Entwicklung des Wrappers kann allerdings nicht komplett entkoppelt von der Clientanwendung stattfinden. Deswegen findet sie in der Arbeit in Auszügen Erwähnung, um einerseits Sachverhalte besser zu erläutern bzw. darzustellen und andererseits um ein realisierbares, praxistaugliches Gesamtsystem zu erhalten. Der Arbeit wird allerdings ein einfaches Werkzeug in Form einer Windows-Forms-Anwendung zu Demonstrations- und Testzwecken der Schnittstelle beigefügt.

Das Sequenzdiagramm im Bild 2.4 soll vereinfacht an einem abstrakten Beispiel einen Überblick über den Ablauf der Kommunikation zwischen den verschiedenen Instanzen des gesamten Systems abbilden. Im Beispiel werden dem Nutzer Daten angezeigt, wel-

che er durch eine Eingabe verändert. Dazu startet er als erstes die Clientapplikation (1). Dort werden ihm dann Daten angezeigt (6), welche mittels Request(2)/Response(5) über den Wrapper von der Datenbank abgerufen wurden (3) (4). Der Nutzer möchte diese Daten nun verändern und macht diesbezüglich eine Eingabe (7). Diese findet wiederum über Request(8)/Response(11) der Clientapplikation den Weg zum Wrapper und schließlich zur Datenbank (9) (10). Dem Nutzer werden anschließend die geänderten Daten präsentiert (12). Der Nutzer könnte nun weitere Änderungen vornehmen, sich andere Informationen anzeigen lassen oder die Applikation beenden.

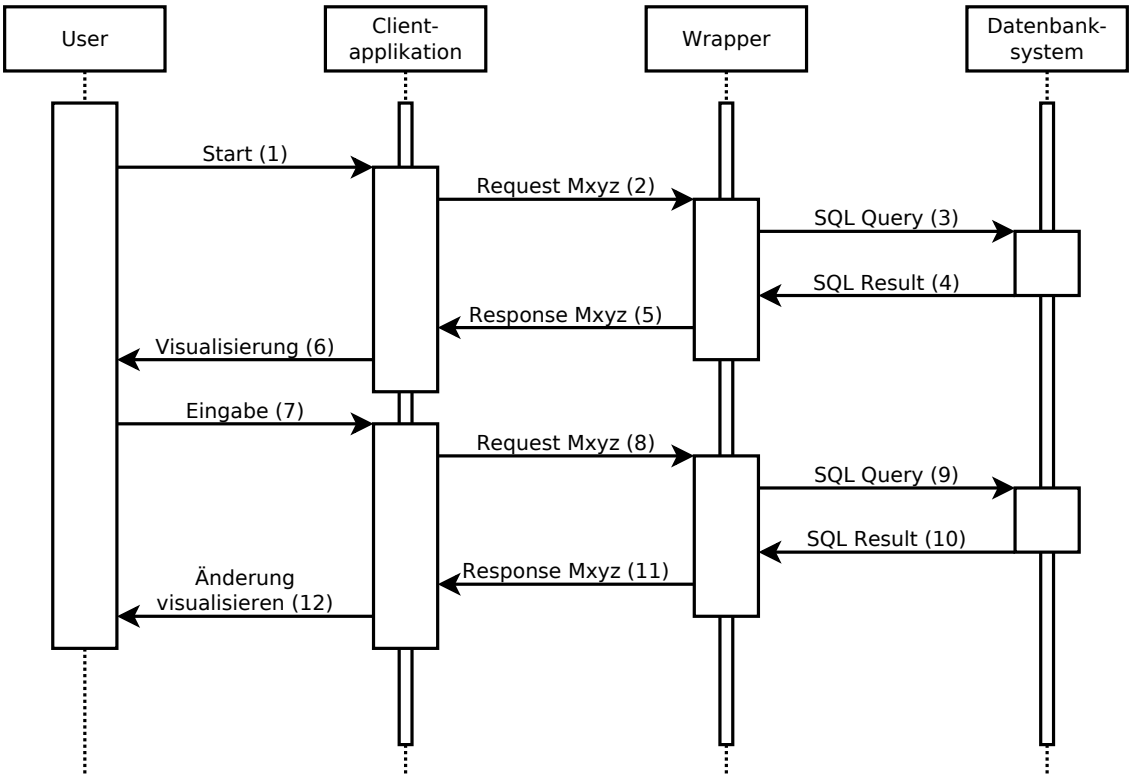

Abbildung 2.4: Sequenzdiagramm zeigt ein verallgemeinertes Beispiel der Kommunikation über alle Systeme

2.2 Anwendungsdomäne

2.2.1 Entwicklungswerkzeug Domain-Driven-Development

Das Ziel einer Software ist die Arbeit in einem gewählten Kontext zu erleichtern bzw. zu unterstützen. Dieser Kontext wird auch als Anwendungs- oder Problemdomäne bezeichnet. Er schafft einen Raum und legt dessen Objekte, Beziehungen sowie Begrifflichkeiten fest. Er bildet bei Enterprise Software (dt. Großkunden-/Unternehmens-anwendung) die fachspezifische Geschäftslogik der Aufgabenstellung ab. Beispielhaft wird aus einer Person eine Entität namens „Mitarbeiter“, aus einer Buchung wird ein Aggregat mit z.B.

den Objekten „Zeitstempel“, „Personalnummer“ und „Job“.[3] Es entsteht somit ein Modell.

In der Softwareentwicklung gibt es eine Herangehensweise, welche genau mit diesem Modell arbeitet. Sie wird Domain-Driven-Development (*DDD*) genannt. Diese Verfahrensweise wird in vielen Entwicklungen großer Projekte sinnvoll und mit Erfolg eingesetzt.[4] Allerdings gibt es ebenfalls viele Anwendungsfälle, wo ein Einsatz nicht zweckmäßig ist. Der zu betrachtende Anwendungsfall scheint auf den ersten Blick genau eine solche Anwendungsdomäne zu besitzen. Wie erwähnt könnten Entitäten und Wertobjekte gefunden sowie Aggregate erstellt werden. Ein Modell, welches die Geschäftslogik abbildet, würde entstehen.

Ein Verständnis für genau dieses Modell, sprich für die Anwendungsdomäne, und dessen Logik ist immanent wichtig. Aber ob diese Modell als zentrales Steuerungsorgan der Entwicklung zu verstehen ist, ist in Frage zu stellen. Um eine Antwort zu finden, ist der Hauptzweck der Applikation zu beleuchten. Ist der Hauptzweck, genau in dieser Anwendungsdomäne zu agieren und außerdem zu präsentieren, dann wäre *DDD* ein dienliches Werkzeug. Ist allerdings der Hauptzweck ein anderer, und genau dies ist hier der Fall, dann sollte *DDD* nicht angewendet werden.[5] Der Wrapper, wie bereits erwähnt, soll nur als Vermittler, quasi als Übersetzer mit geringer Intelligenz fungieren. Um bei dem Bild des Übersetzers zu bleiben, soll auch der Wrapper als Sprachmittler agieren. Dabei ist es allerdings nicht notwendig den zu übersetzenden Text zu interpretieren. Ein weiterer wichtiger Grund für die getroffene Entscheidung gegen *DDD* ist die Art der Informationsübermittlung. Diese soll nicht objektorientiert wie im Sinne einer *COBRA*-Middleware sein, sondern nachrichtenorientiert, wie in der Anforderungsanalyse definiert. Die Nachrichten selbst sowie deren Übermittlung bilden sozusagen die eigentliche Problemdomäne des Wrappers. Dies wird in Kapitel 4.1.4 diskutiert.

Trotzdem ist es wichtig die Geschäftslogik sowie dessen Sprache zu verstehen, deswegen folgt im Anschluss eine Erläuterung dazu. Die Domän gesteuerte Entwicklung entpuppt sich aus den genannten Gründen nicht als Patentlösung. Allerdings bleibt sie ein starkes Entwicklungswerkzeug.

2.2.2 Geschäftlogik/-sprache

Um die Anwendungsdomäne und dessen Sprache besser zu verstehen, werden die einzelnen Elemente kurz erläutert. Die entstandene Aufzählung kann als Glossar für diese Arbeit genutzt werden, da viele sachbezogenen Thematiken sich dieses Wortschaftes bedienen. Das Verzeichnis ist im Anhang B.1 zu finden.

[3] Vgl. Ghadir 2014
[4] Vgl. Wikipedia 2014, „Domain Driven Development“
[5] Vgl. Rebenich 2013

2.3 Anforderungsanalyse

Um die Arbeitsaufgabe genauer zu spezifizieren bedarf es einer Anforderungsanalyse. Diese Analyse spezifiziert das zu entstehende System bzw. legt dessen Randbedingungen im Vorfeld fest. Gemessen an diesen Randbedingungen kann das zu entstehende System schon von Beginn des Entwurfs bis hin zur Integration in den Nutzungsalltag bewertet werden. Dies ermöglicht eine Früherkennung von Schwachstellen sowie deren Beseitigung.

Um die Anforderungen an ein System zu finden, müssen dessen Stakeholder identifiziert werden. Als Stakeholder werden ein oder mehrere Personen bezeichnet, „die ein berechtigtes Interesse am Verlauf oder Ergebnis eines Prozesses oder Projektes“ haben. [6] Im konkreten Fall sind dies der Auftraggeber, die Nutzer, die System-Administratoren sowie mögliche Software-Entwickler für Drittanwendungen. Diese vier Personen(-gruppen) haben Wünsche und Erwartungen an das entstehende System. Die Visionen aller Einzelnen wachsen in iterativen Workshops (dt. Zusammenkunft einer Arbeitsgruppe) zu einer gemeinsamen Basis zusammen. Das Ergebnis dieser Zusammentreffen mündet im Pflichtenheft der Stakeholder. Aus dem Pflichtenheft werden anschließend die Anforderungen extrahiert. Bezugnehmend auf das Kano-Modell, müssen diese explizit erwarteten Systemeigenschaften um die implizit erwarteten ergänzt werden.[7] Die folgenden beiden Auflistungen zeigen die allgemeinen sowie funktionalen Anforderungen an das entstehende System als Ergebnis der durchgeführten Workshops.

Allgemeine Anforderungen:

- Nutzerverwaltung der Clientapplikation (Authentifizierung am System, Rechtemanagement)
- Nachrichten basierte Kommunikation zwischen Wrapper und Clientapplikation
- Entgegennahme mehrerer Anfragen aus mehreren Clientapplikationen (Multitasking)
- schnelle Reaktionszeit bis die Antwort bei der Clientapplikation eintrifft
- kein automatisches Ausloggen einer Clientapplikation, sofern diese nicht ausdrücklich beendet wird
- verschlüsselte Verbindung zwischen Wrapper und Clientanwendung
- verschlüsselte Ablage von Passwörtern
- lauffähig unter Windows 7
- Nutzung der vorhandenen IT-Struktur
- einfach erweiterbar

[6] Wikipedia 2014, „Stakeholder“
[7] Schäfer 2010, S.320 f.

Funktionale Anforderungen:

- Anzeige vom eigenem MA-Konto (Urlaub + Überstunden)
- Vorgesetzer kann MA-konto seiner MA einsehen
- Lesen erfolgter Buchungen vom MA
- Anzeige aktuell gebuchter Jobs des MA
- Anlegen von Jobfavoriten mit Speicherfunktion für jeden MA separat
- Projektauswahl über Rückmeldenummer, Kunde, Zeichnungsnummer
- Projektauswahl nur von offenen Jobs inkl. Suchmaske
- Projektauswahl durch Job-Favoritenliste
- Nachträgliches Buchen auf einen Job nur mit Genehmigung eines festzulegenden Genehmigers
- Unterbrechung eines Auftrages durch Angabe einer Zeit bzw. Buchen auf einen anderen Auftrag
- Anzeige der bereits gebuchten Stunden einer Person auf einen bestimmten Job
- Anzeige der bereits gebuchten Gesamtstunden einer AFO auf einen bestimmten Job
- Buchung von Dienstbeginn/-ende ($\rightarrow$ Dienstbuchung) nur mit Genehmigung eine festzulegenden Genehmigers
- Genehmiger soll Vertretermöglichkeit haben
- Möglichkeit der Sperrung von Aufträgen für Buchungen (geschlossene Jobs $\rightarrow$ Schnittstelle Proalpha)
- Beantragung von Urlaub

Die Anforderungen wurden auf deren Machbarkeit geprüft und für realisierbar eingeschätzt. Auf die Erstellung eines Lastenheft wird verzichtet.

2.4 Anwendungsfälle

Der Nutzer des Gesamtsystems, ergo der Clientapplikation, des Wrappers und der Datenbank, ist der Mitarbeiter des Unternehmens. Für ihn ist zwar nur die Clientapplikation sichtbar und er interagiert nur mit dieser, aber dessen Eingaben bzw. Bedienhandlungen haben Konsequenzen auf das gesamte System. Mit anderen Worten ist die Art und Weise seiner Bedienhandlungen, folglich sein Workflow (dt. Arbeitsablauf), für alle Teilsysteme wichtig. Einen Workflow des Nutzers kann man als Anwendungsfall verstehen. Es stecken sehr viele dieser Anwendungsfälle in den funktionalen Anforderungen aus Kapitel 2.3. Die Anwendungsfälle definieren im Großen und Ganzen den Nachrichteninhalt der zwischen Wrapper und Clientapplikation ausgetauscht werden muss. Der konkrete Inhalt der Nachrichten wird in Kapitel 4.1.4 erörtert, die Anwendungsfälle selbst sind Gegenstand dieses Kapitels.

2.4.1 Akteure und ihre Rollen

Um die Anwendungsfälle des Nutzers beleuchten zu können, muss zuerst der Nutzer selbst beschrieben werden. Ein Nutzer einer Software besitzt i. d. R. mehrere Handlungszwänge, welche sich z. T. inhaltlich stark unterscheiden. Er kann die Software in ihrer eigentlichen Funktionalität als Schnittstelle zum bestehenden *ERP*-System verstehen und nutzen, er kann aber z. B. auch die Software administrieren wollen. Diese unterschiedlichen Handlungszwänge kann man als Rollen einzelner Akteure definieren. Für die Clientapplikation gibt es sechs verschiedene Akteure, die ein Nutzer verkörpern kann. Diese sind mit ihren Funktionen und Handlungszwängen in Tabelle 2.1 aufgeführt.

Akteur	Funktion	Handlungszwang	Beispiel
Mitarbeiter	klassischer Benutzer	Nutzer möchte auf das *ERP*-System zugreifen.	Mitarbeiter bucht sich auf einen Job oder beantragt Urlaub.
Vorgesetzter	Abbildung der Unternehmenshierarchie	Nutzer möchte auf Informationen über die ihm untergeordneten Mitarbeiter zugreifen.	Vorgesetzter sieht Mitarbeiterkonten seiner untergeordneten Mitarbeiter ein.
Genehmiger	Abbildung der Unternehmenshierarchie	Nutzer möchte auf Informationen über die ihm untergeordneten Mitarbeiter zugreifen.	Genehmiger bewilligt oder lehnt Genehmigungsabfragen seiner untergeordneten Mitarbeiter ab.
Genehmigungsverwalter	Abbildung der Unternehmenshierarchie	Nutzer möchte die bewilligten Genehmigungsentscheidungen in das ERP-System einpflegen.	Genehmigungsverwalter pflegt einen Urlaubsantrag in das System ein.
Jobverwalter	Systempflege	Nutzer möchte den Jobbestand einsehen oder aktualisieren.	Jobverwalter löst eine Import des aktuellen Jobbestands aus.
Nutzerverwalter	Administration	Nutzer möchte andere Nutzer hinzufügen, bearbeiten oder entfernen.	Nutzerverwalter richtet neuen Mitarbeiter ein.

Tabelle 2.1: Übersicht Akteure und ihre Funktionen sowie Handlungszwänge

Der fehlende Akteur „Vertreter Genehmiger“ ist dem Akteur „Genehmiger“ gleichzusetzen, allerdings besitzt der Nutzer dieses Recht i. d. R. nur temporär. Weiterhin ist zu beachten, dass ein Nutzer mehrere Rechte besitzen kann. Abbildung 2.5 zeigt ein passendes Beispiel aus Sicht des Mitarbeiters mit der Identifikationsnummer 3 (ID). Er ist als Vorgesetzter von den Mitarbeitern mit der ID 2, 4, 7 sowie 8 administriert und kann z. B. deren Mitarbeiterkonten einsehen. Gleichzeitig ist er berechtigt Genehmigungsan-

frage von den Mitarbeitern mit der ID 4, 7 und 9 zu bearbeiten. Trotz alledem bleibt er immer ein gewöhnlicher Mitarbeiter und kann sich entsprechend seiner Projekttätigkeit buchen oder eigene Genehmigungsanfragen stellen.

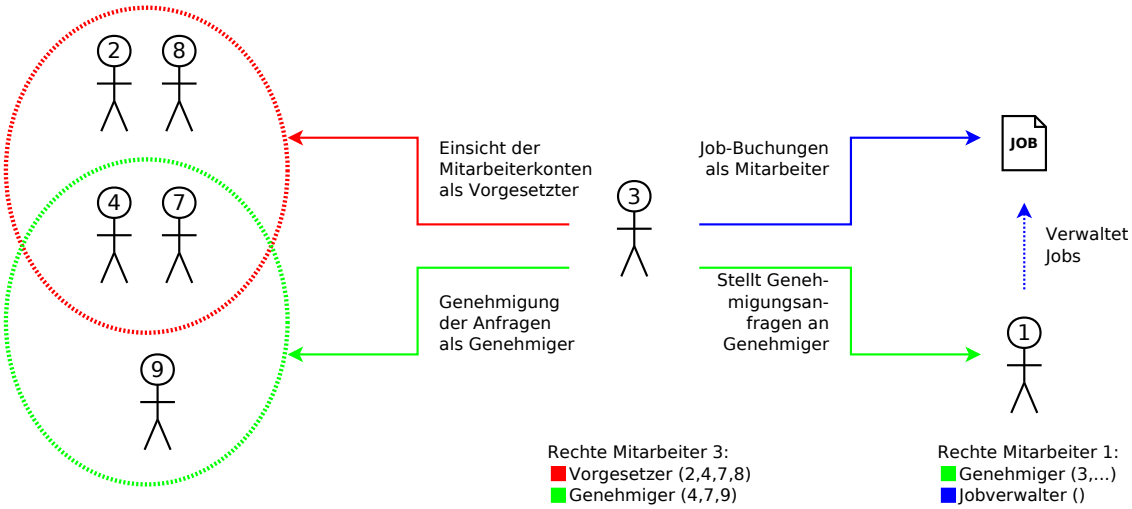

Abbildung 2.5: Beispielhafter Nutzer, welcher mehrere Akteure vertritt, besitzt unterschiedliche Rechte und nutzt diese z. B. in Bezug auf Genehmigungen

2.4.2 Arbeitsabläufe

In der Abbildung 2.5 sind schon erste Handlungen zu erkennen, welche der Nutzer mittels der Clientapplikation durchführen will. Für jede dieser Handlungen sind mehrere Einzelaktivitäten notwendig. Die gezielte Aneinanderreihung von Aktivitäten bildet einen Arbeitsablauf ab.

Ein Teil dieser Einzelaktivitäten bildet das Herzstück der Kommunikation zwischen Clientapplikation und Wrapper. Am Beispiel "Buchung absetzen“ in Abbildung 2.6 erkennt man die unterschiedlichen Einzelaktivitäten. Der Programmablaufplan zeigt, dass neben Operationen (Rechteck) der Nutzer Entscheidungen (Raute) treffen muss, um sein Ziel zu erreichen (Oval). Dabei gibt es eine Teilmenge von Aktivitäten (grün), wo der Nutzer selbst aktiv wird. Die zweite Teilmenge (rot) zeigt Aktivitäten, wo die Clientanwendung mit dem Wrapper kommuniziert. Die dritte Teilmenge (blau) beinhaltet Verarbeitungsschritte der Clientanwendung, welche in diesem Kontext eine untergeordnete Rolle spielen.

Es zeigt sich also, dass für die Wrapperanwendung nicht alle Einzelschritte relevant sind, sondern nur die rot gekennzeichneten. Außerdem ist zu bedenken, dass in den verschiedenen Handlungen einzelne Aktivitäten wiederholt auftreten. Je nach Akteur gibt es unterschiedliche Handlungen mit verschiedenen Arbeitsabläufen.

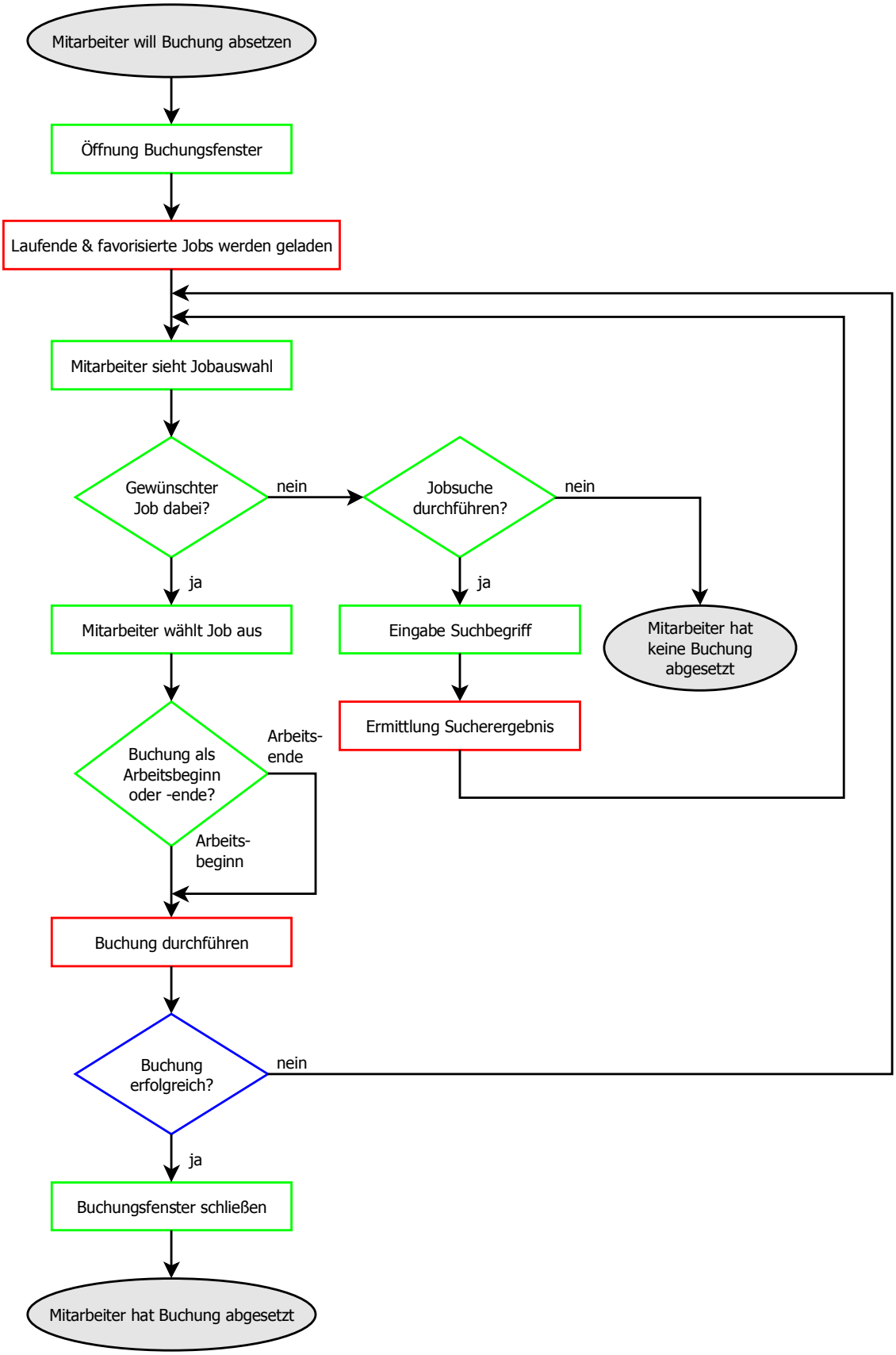

Abbildung 2.6: Ablaufplan für Beispiel „Buchung absetzen“ mit eingefärbten Einzelaktivitäten der unterschiedlichen Teilmengen

Im Anhang A.1 sind die möglichen Handlungen der Akteure aufgelistet. Beispielhaft soll hier der Akteur Nutzerverwalter herausgegriffen werden, welcher fünf Handlungszwänge besitzt. Daraus leiten sich die Aktivitäten für den Wrapper ab, wie das Usecase-Diagramm in Abbildung 2.7 zeigt.

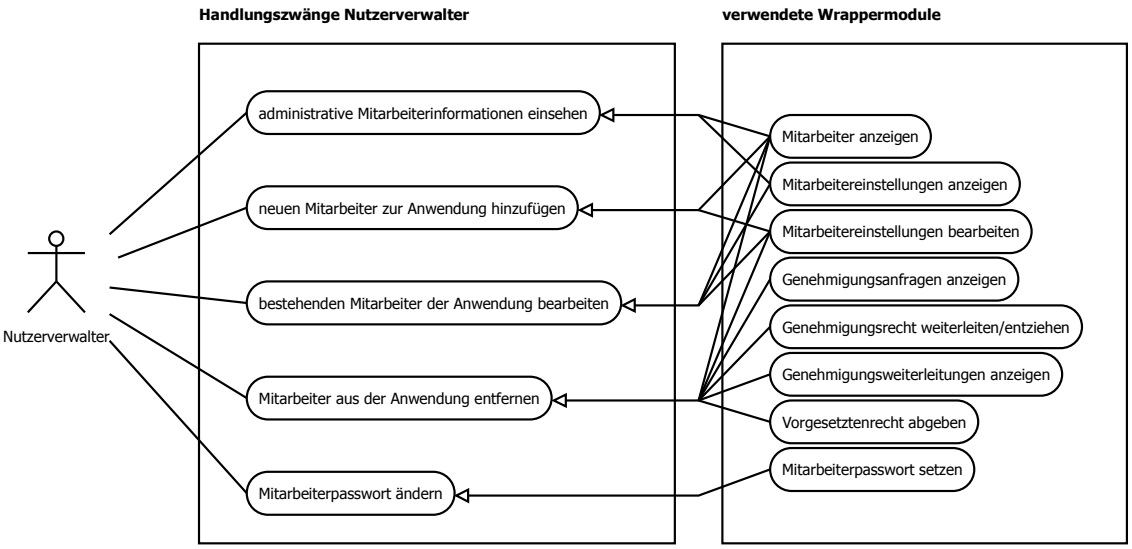

Abbildung 2.7: Handlungszwänge des Nutzerverwalters

Die letzten beiden Beispiele zeigen den Weg, der gegangen wurde um die relevanten Einzelaktiväten für den Wrapper zu ermitteln. Eine Auflistung dieser ist im Anhang A.3 zu finden. Diese Liste bildet die Grundlage für den Entwurf der Kommunikationsinhalte. Um zu definieren, welche Informationen zwischen Clientapplikation und Wrapper ausgetauscht werden können, ist vorher der Informationsbestand zu sichten. Die nächste Aufgabe ist also, das bestehende Datenbanksystem dahingehend zu prüfen, welche Informationen vorliegen, wie diese strukturiert, typisiert und miteinander verknüpft sind.

3 Relationale Datenbanken

3.1 AVERO-Datenbank (Bestand)

Die bestehende Datenbasis „Avero“ ist ein Microsoft SQL Datenbanksystem bestehend aus ca. 150 Tabellen mit jeweils bis zu 90 Attributen bzw. teilweise über 350.000 Einträgen. Neben umfangreichen Tabellen gibt es andere ohne Einträge. Dies liegt an der Modularität des Systems, wobei einige Module beim Kauf nicht erworben wurden und somit ohne Funktion sind. Zum speziellen Datenbanksystem gibt es keinerlei Dokumentation. Man kann die Informationen somit nur anhand ihres Inhalts sowie deren Platzierung mit Tabellen- und Spaltennamen bewerten. Das bestehende System besitzt weder Beziehungen zwischen den Tabellen mittels Fremdschlüssel noch ein ordnendes Element in Form eines Primärschlüssels in den Tabellen selbst. Auch wenn die Schlüsselfunktionalität bei der Entwicklung keine Beachtung fand, so sind diese inhaltlich dennoch vorhanden. Ziel diese Kapitel ist es, die bestehende Datenbasis unter den Gesichtspunkten des zu entwickelnden Systems zu sichten und Beziehungen abzuleiten. Weiterhin sind die Zugriffe der ursprünglichen Eingabe-/Verarbeitungssoftware zu analysieren und aufzubereiten. Das hierfür verwendete Werkzeug wird ebenfalls kurz vorgestellt.

Zur Analyse und für spätere Testzwecke wurde das gesamte Datenbanksystem im Vorfeld kopiert. Die Kopie wurde so eingerichtet, dass darauf wie auf das Original zugegriffen werden kann. Um den Hardwareaufwand diesbezüglich so gering wie möglich zuhalten, wurden Mittel der Virtualisierung eingesetzt. Der damit einhergehende Performance-Verlust ist für die Analyse sowie Tests akzeptabel. Durch das Umleiten der Anfragen mittels Routing konnte nun von einem gewöhnlichen Host (dt. Arbeitsrechner) auf die Testumgebung so zugegriffen werden, wie auf den Originalserver.

Mit dem „SQL Management Studio“[8] der Firma Microsoft können Datenbanken erstellt, bearbeitet und dargestellt werden. Die Software erlaubt somit das Recherchieren in den Daten. Dabei erwiesen sich drei Tabellen für die Stamm- und Bestandsdaten als relevant. Die Tabelle „EmployeeRecord“ enthält Informationen zum Mitarbeiter sowie dessen Name, Personal- oder Badgenummer sowie Bestandsdaten wie dessen Resturlaub- und Überstundenkonto. Die Stammdaten zu den Projekten befinden sich in den beiden Tabellen „Jobs“ und „Operations“. Die erste Tabelle ist nach Rückmeldenummern geordnet und beinhaltet die diesbezüglichen Informationen zu Projektbezeichnung, Kundenname, Auftrags- und Zeichnungsnummer. Eine Rückmeldenummer kann durch Tätigkeiten (=AFO-Nummern) weiter untergliedert werden. Diese Informationen befinden sich in der zweiten Tabelle. Außerdem sind darin die Bestandsdaten zu den Tätigkeiten

[8] Microsoft - SQL Server Management Studio

respektive den Jobs gespeichert. Abbildung 3.1 zeigt alle drei Tabellen und eine Auszug ihre Attribute.

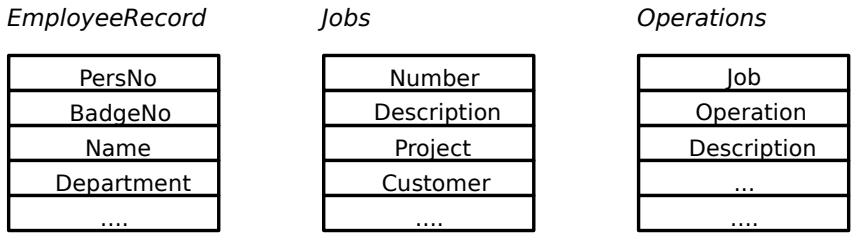

Abbildung 3.1: Auszug aus den Tabellen „EmployeeRecord“„Jobs“ und „Operations“ der Avero Datenbank

Die Job-bezogenen Bewegungsdaten sind ebenfalls auf verschiedene Tabellen verteilt. Diese haben einen unterschiedlichen Verarbeitungsgrad sowie zeitlichen Rahmen. Es gibt die Tabellen „PCTBooking“ und „TMOnline“ für die kurzfristige Erfassung der unverarbeiteten Bewegungsdaten von Buchungen. Die langfristige Speicherung erfolgt hingegen in der Tabelle „TMData“. Werden die Daten durch den Hintergrunddienst verarbeitet, dann wird das Ergebnis kurzfristig in „TMBooking“ und langfristig in „Booking“ gespeichert. Abbildung 3.2 zeigt eine Übersicht über den Verarbeitungsgrad sowie den zeitlichen Rahmen.

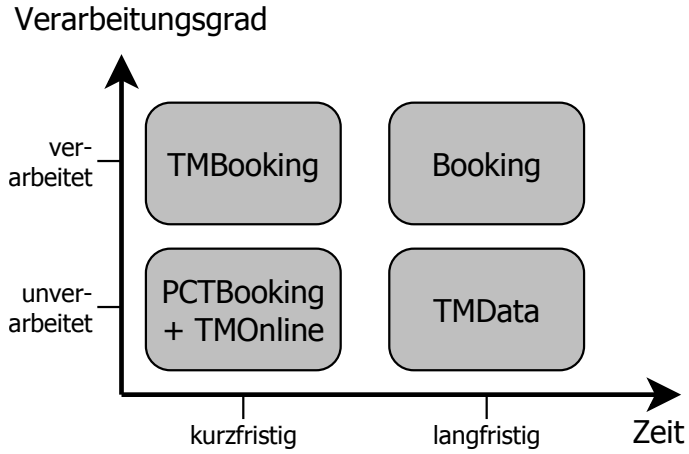

Abbildung 3.2: Klassifizierung der Tabellen mit Job-bezogenen Bewegungsdaten

Wie schon angedeutet gibt es einen gleichnamigen Hintergrunddienst namens „Avero“. Diese Applikation verarbeitet Daten. Anhand der obig gezeigten Tabellenstruktur bedeutet dies, dass er die Bewegungsrohdaten aus den Tabellen „PCTBooking“ und „TMOnline“ verarbeitet und die Bestandsdaten entsprechend anpasst oder ergänzt. Außerdem erfolgt die Umspeicherung in eine der langfristigen Tabellen. Die beiden Tabellen dienen also nur als Container für unbearbeitete Aufgaben, die durch den Hintergrunddienst noch zu verarbeiten sind. Um die Arbeitsweise dieser Verarbeitungseinheit sichtbar zu machen, wurde der „SQL-Profiler“[9] genutzt. Dieses Werkzeug des Management Stu-

[9] Microsoft - SQL Server Profiler

dios ermöglicht die Aufzeichnung sämtlichen Datenverkehrs von bzw. zum Datenbanksystem. Im Anschluss können die Datenströme entsprechend gefiltert und analysiert werden. Nachfolgend ist ein solcher Datenstrom abgebildet.

```
declare @p1 int
set @p1=5
exec sp_prepexec @p1 output,NULL,N'INSERT INTO TMOnline
( [Booking], [Date], [Time], [BadgeNo], [PersNo] )
VALUES ( ''19.07.2013|08:15:00|-1/1|001|PCTL000013006||1|
30714'', ''20130719'', ''08:15:00'', ''30714'', ''000043'' )'
select @p1
```

Der Datenstrom zeigt die Verwendung einer gespeicherten Systemprozedur namens „sp_prepexec“. Solche Systemprozeduren bilden die Funktionalität einer Datenbank-API ab. Es kann allerdings nicht direkt auf sie zugriffen werden, vielmehr sind sie für die interne Verwendung z. B. in Treibern vorgesehen. Eigene Prozeduren kommen allerdings nicht zum Einsatz.

3.2 Wrapper-Datenbank (Erweiterung)

Um nun weitere Funktionen in das System zu integrieren muss die Datenbasis erweitert werden. Da das bestehende Datenbanksystem nicht in dessen Struktur verändert werden soll, muss eine weitere Datenbank hinzugefügt werden - die Wrapper-Datenbank. Sie soll dem Wrapper selbst sowie der Clientapplikation einen Informationshaushalt bieten. Nachfolgende Funktionalitäten benötigen hierfür eine zusätzliche Datenablage.

- Nutzerverwaltung mit An-/Abmeldung, Vertreterregelung
- Jobverwaltung mit Status und Favoriten
- Genehmigungsverwaltung

Nachfolgend sollen die entstandenen zusätzlichen Entitäten kurz vorgestellt werden. In den ER-Modellen sind die Primärschlüssel hervorgehoben sowie die Beziehungen zwischen den Entitäten mit Doppelpfeil-Linien gekennzeichnet. Dabei wurde die Darstellungsform nach Zehnder[10] gewählt und in die vier Assoziationstypen aus Tabelle 3.1 unterschieden.

3.2.1 Nutzerverwaltung

Personen werden anhand ihrer Personalnummer identifiziert, deswegen soll dieses Attribut auch als Primärschlüssel fungieren. Zur Anmeldung wird weiterhin ein Passwort

[10] Vgl. Zehnder 1998

Beziehung	Assoziation	Anzahl verbundener Tupel
1	einfach	1
C	konditionell	0 oder 1
M	multiple	≥ 1
MC	multiple-konditionell	≥ 0

Tabelle 3.1: Assoziationstypen nach Zehnder

benötigt, welche mehrfach MD5 gehasht abgelegt wird. Hat der Nutzer der Clientapplikation sich richtig authentifiziert, so wird anschließend mittels eines Tokens zwischen Client und Vermittler kommuniziert. Weiterhin besitzt ein Nutzer i. d. R. einen Vorgesetzten sowie einen Genehmiger. Diese können die gleiche oder zwei unterschiedliche Personen sein. Die verschiedene Akteure benötigen unterschiedliche Bedienberechtigungen. Solche Zusatzberechtigungen sind z. B. Nutzerverwalter bzw. Jobverwalter. Ein Nutzer erhält weitere Berechtigungen, wenn er als Vorgesetzter oder Genehmiger bei anderen Personen eingesetzt wird. Integriert man noch die Genehmigungsvertreter-Funktionalität, so erhält man eine Datenstruktur wie in Abbildung 3.3 dargestellt.

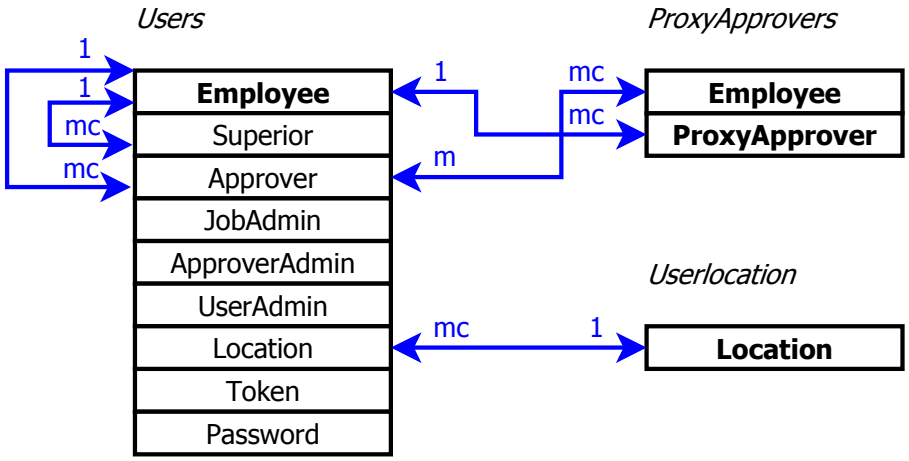

Abbildung 3.3: ER-Modell mit Tabellenstruktur für Nutzerverwaltung

Bei dem abgebildeten ER-Modell sind die rekursiven Verknüpfungen zwischen „Employee“ und „Approver“ bzw. „Superior“ in der Tabelle „Users“ auffällig. Derartige Beziehungen können auf verschiedene Herangehensweisen in der Datenbank realisiert werden. In Tabelle 3.2 sollen drei davon kurz mit ihren Vor- und Nachteilen vorgestellt werden.

Der verwendete SQL Server basiert auf der Version 2005, somit steht die Variante mit dem „HIERARCHYID“-Datentyp nicht zur Verfügung. Die Unterschiede zwischen Variante 1 und 2 sind gering. Für eine Selektion ist zu entscheiden, ob der Mehraufwand durch die Variante 1 gerechtfertigt ist. Im Fall der Genehmigungsvertreter „ProxyApprover“ ist ein Ausgliedern in eine separate Tabelle zweckmäßig, da es ohnehin mehrere Vertreter pro Mitarbeiter geben kann. Bei den rekursiven Verknüpfungen mit „Approver“ bzw. „Superior“ gibt es keine funktionalen Gründe auszugliedern, zudem würde eine Realisierung in einer Tabelle die Zugehörigkeit verdeutlichen. Deswegen sollen diese

Realisierung	Vorteile	Nachteile
separate Tabelle	keine rekursive Verknüpfung	mehr Tabellen
	keine NULL-Elemente	Hierarchieabbildung aufwendig
		tabellenübergreifende SQL-Befehle
Datentyp „HIERARCHYID“	SQL-Befehle für Hierarchieabbildungen	erst ab SQL Server 2008 verfügbar
	kein tabellenübergreifender SQL-Befehl	
Fremdschlüssel	keine weiteren Tabellen	Hierarchieabbildung aufwendig
	kein tabellenübergreifender SQL-Befehl	

Tabelle 3.2: Umsetzung von Rekursionen im Datenbanksystem

beiden Rekursionen jeweils mittels Fremdschlüssel umgesetzt werden. Der nachfolgende SQL-Befehl zeigt die Verwendung der Einschränkung am Beispiel „Approver“.

```
ALTER TABLE Wrapper.dbo.Users
ADD CONSTRAINT FK_Users_Users_Approver
FOREIGN KEY(Approver)
REFERENCES Wrapper.dbo.Users (Employee)
```

3.2.2 Jobverwaltung als Data-Warehouse

Eine Grundregel bei dem Entwurf von Datenbanksystem ist, keine doppelte Datenhaltung zu führen. Dieses Konzept gilt natürlich auch bei der Verwendung von mehreren Datenbanksystemen, wie es hier der Fall ist. Allerdings kann dies bei komplexen Anfragen über mehrere Systeme und Tabellen hinweg zu erheblichen Verzögerungen führen. In diesen Fällen greift man daher meist auf ein „Data-Warehouse“ zurück. Dies ist eine Datenbank, welche aus mehreren, verteilten Datenbestände eine gemeinsame Sicht auf diese bildet. Für die Job-Information ist, neben den zwei bereits vorhandenen Tabellen, eine weitere Tabelle notwendig. Abfragen würden sehr komplex bzw. aufgrund der großen Anzahl der Wertpaare auch zeitintensiv sein. Deswegen soll an dieser Stelle der „Data-Warehouse“-Prinzip aufgriffen werden. Die Jobinformationen werden dazu in einer Tabelle der Wrapper-Datenbank gebündelt und um zusätzliche ergänzt, wie Abbildung 3.4 zeigt.

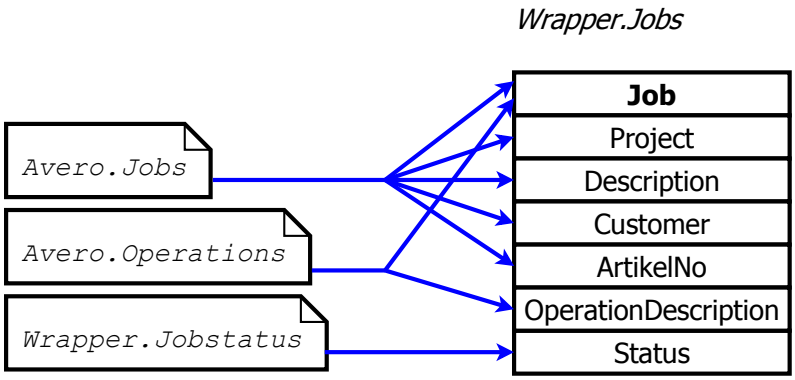

Abbildung 3.4: Tabelle „Jobs“ der Wrapper-Datenbank als Bündelung nach dem „Data-Warehouse“-Prinzip zur Leistungssteigerung

Diese Tabelle enthält als Primärschlüssel die Jobnummer, welche sich aus Rückmelde- und Arbeitsfolgenummer zusammensetzt. Außerdem sind zusätzliche Angaben wie Projektbezeichnung, Kunde, Tätigkeitsbezeichnung, Jobstatus, Auftrags- sowie Zeichnungsnummer enthalten. Abbildung 3.5 zeigt die diesbezüglichen Tabellen und Beziehungen.

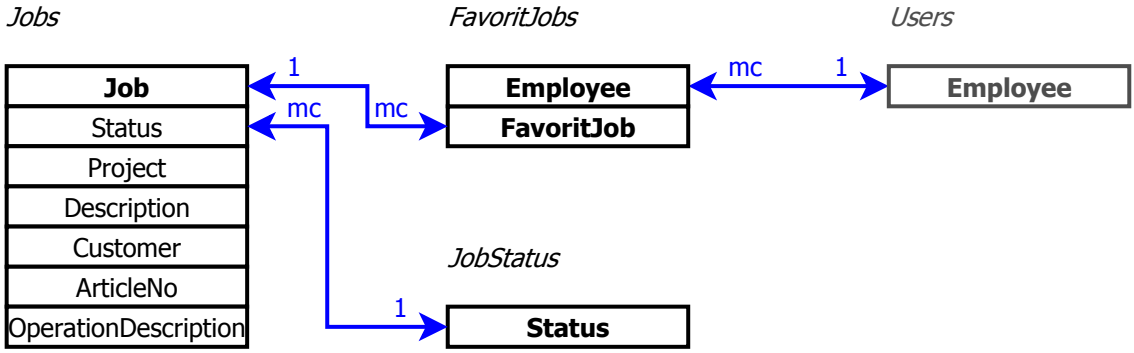

Abbildung 3.5: ER-Modell mit Tabellenstruktur für die Jobverwaltung

Die Tabelle „FavoritJob“ dient der Ablage der mitarbeiterbezogenen Jobfavoriten, wobei jeder Mitarbeiter mehrere Favoriten anlegen kann. Die Tabelle „JobStatus“ definiert die möglichen Jobstatus, z. B. offen oder geschlossen.

3.2.3 Genehmigungsverwaltung

Laut Anforderungsliste gibt es fünf genehmigungspflichtige Tätigkeiten. Diese Tätigkeiten besitzen gleiche sowie unterschiedliche Attribute. Trotzdem sollen sie gemeinsam in einer Tabelle abgespeichert werden, auch wenn einige Attribut-Wert-Kombinationen nie verwendet werden. Primärschlüssel ist eine dezimaler Genehmigungskennung. Weitere gemeinsame Attribute sind der beantragende Mitarbeiter, der Genehmiger, das Buchungsdatum,der Verbucher, die Genehmigungsart und der Genehmigungsstatus. Je nach Genehmigungsart fallen noch weitere Attribute an - Parameter 1 - 4. Die vier

Genehmigungsarten sind nachträgliches Buchen, Dienstbuchungen, Antrag auf Urlaub oder Absetzung von Überstunden. Die Datenstrukturen werden im ER-Modell in Abbildung 3.6 dargestellt.

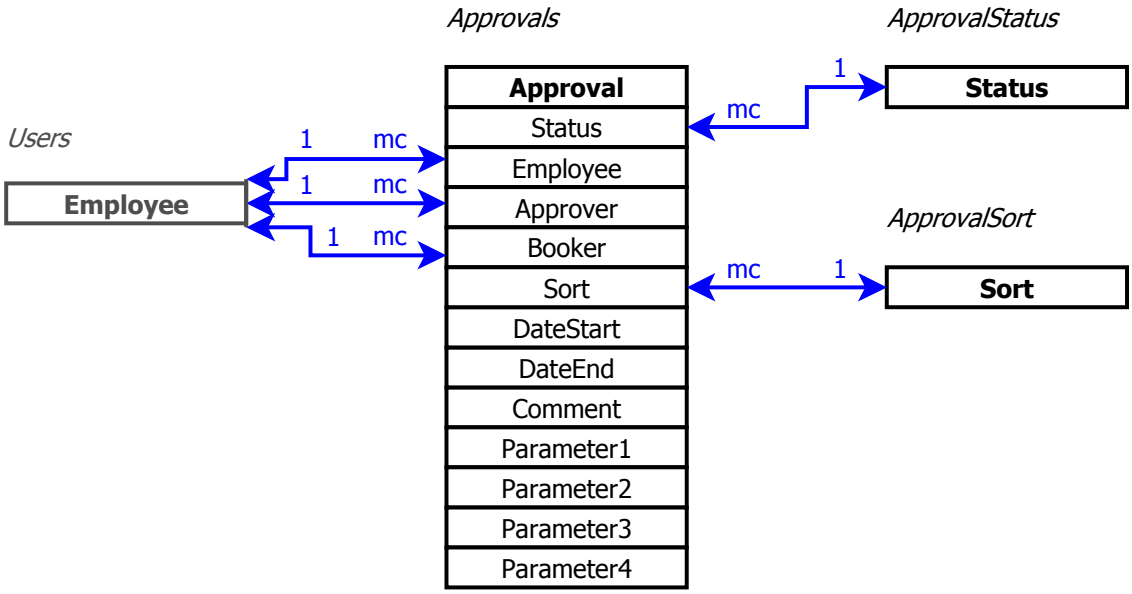

Abbildung 3.6: ER-Modell mit Tabellenstruktur für die Genehmigungsverwaltung

4 Kommunikation

4.1 Nachrichten

4.1.1 Übermittlung

Der Aufbau einer Kommunikation zwischen verteilten Anwendungen geschieht i. d. R. nach dem *OSI*-Modell.[11] Alternativ kann auch das in den 1960er Jahren vom Verteidigungsministerium der Vereinigten Staaten entwickelte *DoD*-Schichtenmodell, welches unter dem gebräuchlichen Namen *TCP*/*IP*-Modell bekannt ist, herangezogen werden. Beide Schichtenmodelle sind in den Schichten 1 - 4 gleich. Nur im Detaillierungsgrad der Anwendungsschicht unterscheiden sie sich. Beim *TCP*/*IP*-Modell sind die Schichten 5 - 7 in der Anwendungsschicht zusammengefasst, das *OSI*-Modell unterscheidet in Sitzungs- (5), Darstellungs- (6) und Anwendungsschicht (7). In den weiteren Ausführungen wird sich auf das *TCP*/*IP*-Modell bezogen. Es sind somit ggf. fünf Schichten festzulegen. Da das entstehende System in die bestehende Netzwerkarchitektur, siehe Abbildung 4.1, der Firma integriert werden soll, sind dessen Informationswege zu nutzen. Durch diese Vorgabe stehen die Schicht 1 und 2 (Ethernet) sowie Schicht 3 (*IP*) des Schichtenmodells fest. In der Schicht 4 ist zwischen *UDP* und *TCP* zu wählen.

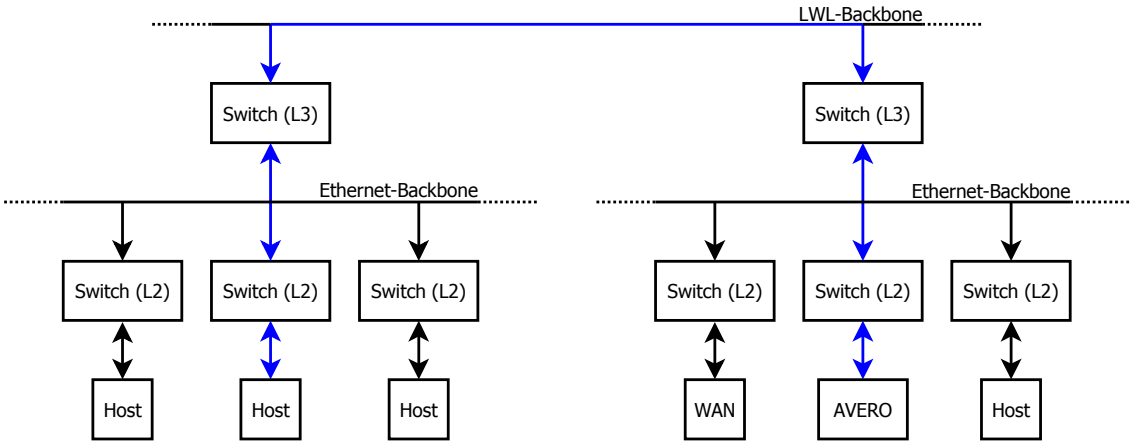

Abbildung 4.1: Bestehende Netzwerk-Infrastruktur mit Kennzeichnung des angestrebten Informationsweges

UDP arbeitet verbindungslos und besitzt nur einen kleinen 64bit-Header. Dadurch weist das Protokoll nur einen geringen Overhead (dt. Verwaltungsdaten) auf und ist für einen datenintensiven sowie zeitkritischen Informationsaustausch prädestiniert.[12] Da das Protokoll verbindungslos arbeitet, eignet es sich für Anwendungen, die Informationen als

[11] Vgl. International Telecommunication Union 1994
[12] Vgl. ITWissen 2014

Broadcasts (dt. Rundruf) versenden. Typische Vertreter hiervon sind *VoIP*- oder Streaming-Konzepte, wie das Internet-Radio. Da bei *UDP* auf eine gesicherte Verbindung verzichtet wird, können allerdings Informationen in der falschen Reihenfolge empfangen werden oder teilweise sogar verloren gehen. Bei den genannten Vertretern ist letzteres durchaus erwünscht, da das Warten auf Sprach- oder Videoinformationen meist nicht dienlich ist. Ein geringer Informationsverlust bringt dabei nur einen geringen Qualitätsverlust mit sich. Bei Anwendungen, wo jede Information wichtig ist, ist dies ein entscheidender Nachteil bzw. in den meisten Fällen ein Ausschlusskriterium für den Einsatz von *UDP*.

Die Alternative *TCP* bietet durch dessen verbindungsorientierte sowie paketvermittelte Übertragungsart einen anderen Lösungsansatz. Dabei greifen verschiedene Konzepte der Transportsicherung und bieten eine Verbindung ohne einen (Netto-)Informationsverlust. Diesem Vorzug stehen ein mehr als doppelt so großer Header und eine damit geringere Nutzdatenlänge entgegen. Dies wirkt sich wiederum negativ auf die reale Übertragungsgeschwindigkeit der Informationen aus. Nichtsdestotrotz, die Kombination aus *TCP* und *IP*, kurz *TCP*/*IP*, bildet „das“ Transportprotokoll der heutigen Technologien. Es wird z. B. bei den Datenübertragungskonzepten des Internets (*HTTP*, *FTP*) oder von Emails (*SMTP*, *POP3*, *IMAP*) eingesetzt.

Für das angestrebte System ist ein Informationsverlust nicht akzeptabel. Weiterhin sind Broadcasts nicht notwendig sowie eine geringe Nutzdatenlänge aufgrund des geringen Informationsaufkommens akzeptabel. Die Entscheidung fällt demzufolge zugunsten von *TCP* in der Transportschicht. Mit einer Implementierung bis Schicht 4 wäre die Übertragung von Nachrichten mittels Sockets (dt. Datensteckdose) bereits möglich. Gleichwohl ist ein Blick in die darüberliegende Schicht ratsam, da unter Einsatz eines geeigneten Anwendungsprotokolls der Programmieraufwand sowie das Fehlerpotenzial reduziert werden können. Nachfolgende Anwendungsprotokolle sind gängige Konzepte.

- *Telnet*: Protokoll für den zeichenorientierten Datenaustausch z. B. für den Fernzugriff auf andere Computersysteme
- *SMTP*, *POP3*, *IMAP*: Spezielle Protokolle für das Empfangen, Versenden und Verwalten von Emails.
- *FTP*: Protokoll zur Übertragung von Dateien.
- *HTTP*, *HTTPS*: Protokoll zur Übertragung von strukturierten Daten, z. B. Webseiten.
- *LDAP*: Protokoll zur Abfrage und Anpassung von Informationen eines Verzeichnis-Servers.
- *NCP*: Protokoll für die Nutzung von Ressourcen wie Drucker oder Speicher im Kontext des Netzbetriebssystem NetWare.
- *OPC UA*: Protokoll zur Übertragung von Maschinendaten zwischen zwei Maschinen.
- *UDS*: Protokoll zur Kommunikation mit Steuergeräten in der Automobilelektronik.

Die Protokolle *LDAP*, *NCP*, *OPC UA* und *UDS* scheiden aufgrund ihrer stark eingegrenzten Einsatzgebiete aus. *FTP* ist ungeeignet, da das Verpacken, Entpacken sowie Ablegen von Dateien unnötig Ressourcen beanspruchen würde. Weiterhin ist ein Transport über die genannten Mail-Protokolle nicht ratsam. Die Mailstruktur bringt zwar geringfügige Vorteile, aber der Aufwand beim Einrichten sowie Administrieren des Mailservers sowie die Komplexität beim Verarbeiten der Mails wäre unverhältnismäßig hoch. Bleiben die Protokolle *Telnet* und *HTTP*/*HTTPS*. *Telnet* bietet die einfache Möglichkeit Nachrichten zeichenorientiert zu übertragen. *HTTP* bzw. *HTTPS* bieten dies ebenfalls an, liefern allerdings noch weitere Funktionalitäten mit, z. B. die Verschlüsselung mittels Zertifikaten. Das Hypertext-Übertragungsprotokoll ist nativ im .NET-Framework enthalten. Der Implementierungsaufwand wäre somit gering. Nachrichten könnten direkt als Nutzdaten im Messagebody (dt. Nachrichtenkörper) übertragen werden. Die *HTTP*-Methoden GET, POST und PUT liefern einfache, aber nützliche Zugriffs- oder Vermittlungsmöglichkeiten auf bzw. von Ressourcen.

Bei der Kommunikation zwischen Clientapplikation und Wrapper kommt aufgrund der genannten Vorteile das Protokoll *HTTP*/*HTTPS* zum Einsatz.

4.1.2 Anwendungsprotokoll *HTTP/HTTPS*

Die Kommunikation zwischen Client und Server erfolgt auf Basis des Hypertext-Übertragungsprotokolls, wie bereits in Kapitel 4.1.1 erörtert. Das Protokoll liegt aktuell in zwei Versionen vor. Die Version *HTTP*/1.0 ist im Standard *RFC* 1946 definiert. 1999 erschien die Erweiterung *HTTP*/1.1 in dem Standard *RFC* 2616, welcher 2014 von den *RFC* 7230...7235 abgelöst wurde. Die neuere Version enthält Verbesserungen z. B. beim Aufbau und Nutzung von persistenten TCP-Verbindungen.[13] Zur Übertragung von Daten eignen sich die drei Methoden „GET“, „POST“ und „PUT“. Dabei dient „GET“ der Anfrage von kleinen und „POST“ von großen Datenmengen. Mittels der Methode „PUT“ können ganze Ressourcen übertragen werden. Eine typische Anfrage hat ca. 300 Zeichen, somit ist die „GET“-Methode in der Protokoll-Version 1.1 ausreichend.

Bei Kommunikation ist Sicherheit ein wichtiger Grundaspekt. Um dies zu verdeutlichen, wurde ein freies Programm zur Analyse von Netzwerk- und Kommunikationsverbindungen verwendet. Mittels einem sogenannten Sniffer (dt. Schnüfler) lässt sich der Datenverkehr zwischen zwei Datenendpunkten mitschneiden und analysieren. Das eingesetzte Programm „Wireshark“[14] verfügt über geeignete Filterfunktion um den nicht relevanten Datenverkehr auszublenden. Nach der Filterung konnte mittels der integrierten Analysewerkzeuge der komplette Datenverkehr zwischen Client- und Serverapplikation im Klartext sichtbar gemacht werden. Dies birgt zwei Risiken. Zum Einen werden über die Serverschnittstelle Informationen, die dem Datenschutz unterliegen, transpor-

[13] Vgl. Heise 2014
[14] Wireshark 2014

tiert. Zum Anderen ist somit ein Versenden von kompromittierten Nachrichten möglich. Anzumerken ist, dass durch die Verwendung von *L2*-Switches und dem damit stark verzweigten physikalischen Aufbau, welcher in Abbildung 4.1 dargestellt ist, ein Mithören erschwert wird.

HTTP liefert mit dem Zusatz „Secure“ (dt. sicher) eine Verschlüsselungstechnik, welche im betrachteten Anwendungsfall implementiert werden soll. Bei *HTTPS* wird das unverschlüsselte Protokoll mit dem *SSL/TLS*-Protokoll chiffriert. Haupteinsatzgebiet ist die Authentifizierung und Verschlüsselung zu bzw. von Webinhalten. Dabei wird zuerst der Server mittels einem digitalen Zertifikat identifiziert. Anschließend wird entweder mit dem Diffie-Hellmann-Verfahren inkl. Perfect Forward Secrecy (dt. perfekte vorwärts gerichtete Geheimhaltung) oder einem asymmetrischen Schlüsselaustausch-Verfahren ein symmetrischer Sitzungsschlüssel ausgehandelt. Alle drei Verfahren sollen kurz vorgestellt werden.

Das Public-Key-Verfahren (dt. öffentliches Schlüsselverfahren) für die Identifikation von Kommunikationsteilnehmern basiert auf Vertrauen. Dieses Vertrauen wird zentral von einer Certification Authority (dt. Zertifizierungsstelle) ausgesprochen. Wird eine Instanz von einer *CA* als vertrauenswürdig eingestuft und somit auch eindeutig identifiziert, erhält sie ein entsprechendes Zertifikat. Die Instanz kann nun auf Grundlage des vorhanden Zertifikats weiteren (Unter-)Instanzen vertrauen aussprechen - und so weiter. Mit einem solchen Zertifikat kann der Ursprung eines Public-Keys bestimmt werden. Die Identifikation einer Instanz erfolgt wie in Abbildung 4.2 dargestellt.[15] Im Vorfeld muss sich der Server bei einer *CA* zertifizieren lassen und der Client muss vertrauenswürdige Stammzertifikate installieren - i. d. R. wird bei der Einrichtung eines Web-Browser eine Vielzahl solcher Zertifikate auf dem Client hinterlegt. Findet eine Kommunikation statt, so wird mittels *SSL/TLS*-Handshake (dt. Flusssteuerung) der Zertifikatsaustausch bewerkstelligt. Nach der Validierung gilt der Server als authentifiziert.

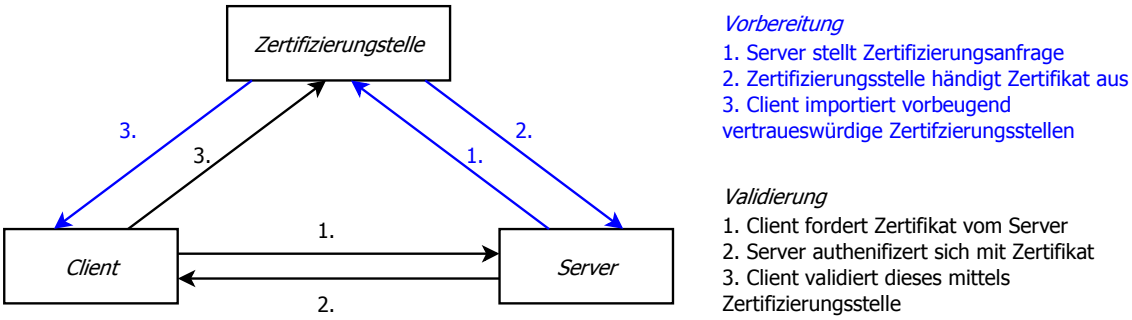

Abbildung 4.2: Identifizierung eines Servers mittels Zertifizierung - Vorbereitung und Validierung

Danach muss ein gemeinsamer (symmetrischer) Sitzungsschlüssel beiden Instanzen bekannt gemacht werden. Dazu wird i. d. R. das Diffie-Hellmann-Verfahren angewendet. Bei diesem Verfahren erfolgt kein physikalischer Sitzungsschlüsselaustausch, son-

[15] Vgl. Schnabel 2014

dern es wird ein gemeinsamer Schlüssel mittels großer Primzahlen und Moduloberechnung ermittelt.[16] Beim asymmetrischen Schlüsselaustausch wird der Sitzungsschlüssel chiffriert übertragen. Dazu verschlüsselt der Client den Sitzungsschlüssel mit dem Public-Key des Servers. Somit kann ausschließlich der Server den Sitzungsschlüssel entschlüsseln. Da die Übertragung des Sitzungsschlüssels eine Gefährdung der Sicherheit darstellt, wird dieses Verfahren nur noch selten angewendet. Allerdings ist der Rechenaufwand gegenüber mathematischen Verfahren deutlich geringer.

Der symmetrische Sitzungsschlüssel, welcher sowohl dem Client wie auch dem Server bekannt ist, wird zum Verschlüsseln des Inhalts verwendet.

4.1.3 Auswahl der Auszeichnungssprache

Damit sich zwei Personen unterhalten können, müssen sie dieselbe Sprache sprechen. Sieht man von der Lautschrift ab, gibt es zwei Faktoren, die eine Sprache von einer anderen meist grundlegend unterscheidet. Dies ist zum einen der Wortschatz und zum anderen die Grammatik. Der Wortschatz bildet dabei den Inhalt und die Grammatik gibt durch die Anordnung der Wörter dem Ganzen eine Bedeutung.[17] Sollen sich nun zwei maschinelle Anwendungen verstehen, so bedarf es ebenfalls einer gemeinsamen Sprache mit übereinstimmendem Wortschatz und Grammatik. Bei automatisierten Lese- und Schreibprozessen ist vor allem die Grammatik von entscheidender Bedeutung. Durch eine sachdienliche Grammatik ist es möglich Daten strukturiert abzulegen. Durch die Strukturierung kommt es beim Zugriff auf die Daten, z. B. bei einer Suchanfrage, zu einer verbesserten Performance, da ggf. nicht der gesamte Datenhaushalt durchsucht werden muss.

Um einer Nachricht eine Struktur aufzuerlegen bedarf es im Vorfeld der Wahl einer geeigneten Auszeichnungssprache. Eine Auszeichnungssprache enthält verschiedene Sprachelemente, welche nach dem Baukasten-Prinzip zusammengestellt werden. Die Urauszeichnungssprache ist die normierte, verallgemeinerte Auszeichnungssprache *SGML*. Allerdings ist sie durch ihr fast 30jähriges Bestehen nicht mehr zeitgemäß und überladen. Nachfolgekonzepte wie *HTML* oder *XML* beruhen auf der gleichen Idee, sind aber deutlich entschlackt. Durch den daraus erzielten Leistungsgewinn eignen sie sich besonders gut für eine ressourcenarme Verarbeitung von großen Datenmengen mit einer angemessenen Geschwindigkeit. *HTML* wird zur Strukturierung von Dokumenten mit Texten, Bildern und Hyperlinks verwendet. Genau solche Inhalte findet man auf Webseiten. Deswegen wird diese Auszeichnungssprache sowie ihre Nachfolger für die Strukturierung von Webinhalten eingesetzt.

XML ist eine Sprache um Daten, welche zwischen zwei Systemen ausgetauscht werden

[16] Vgl. Universität Wuppertal 2012
[17] Vgl. Wikipedia 2014, „Sprache“

sollen, zu strukturieren. Begünstigt wird dies durch eine implementations- und plattformunabhängige Arbeitsweise. Aufgrund der maschinellen Kommunikation wird eine datenzentrierte Strukturierung angestrebt, welche sich in Elemente und deren Unterelemente ähnlich einer Baumstruktur gliedert. Im Unterschied zu *HTML* sind die Tags (dt. Auszeichner) allerdings nicht vordefiniert. Die Interpretation der Tags obliegt voll und ganz der Anwendung. Als Interpretation ist hier das De- bzw. Serialisieren von *XML*-Nachrichten in oder aus Objekte der Anwendung zu verstehen. Um diesen Vorteil zu gewährleisten, muss bei einer solchen Interpretation strenger als bei *HTML* geprüft werden. Ist die *XML*-Nachricht nicht wohlgeformt bzw. durch eine solche Prüfung durchgefallen, so kann keine Deserialisierung stattfinden. Mittels Schemata kann ein ganzes Regelwerk zum Inhalt und zur Struktur selbst definiert werden. Dazu kann der Inhalt des *XML*-Dokuments bzw. deren Elemente beispielsweise beschränkt werden. Die Einhaltung solcher Beschränkungen könnte bereits beim Verifizieren einer neuen Nachricht geprüft werden. Das Beispiel in Abbildung 4.3 zeigt wie so eine Beschränkung aussehen kann.

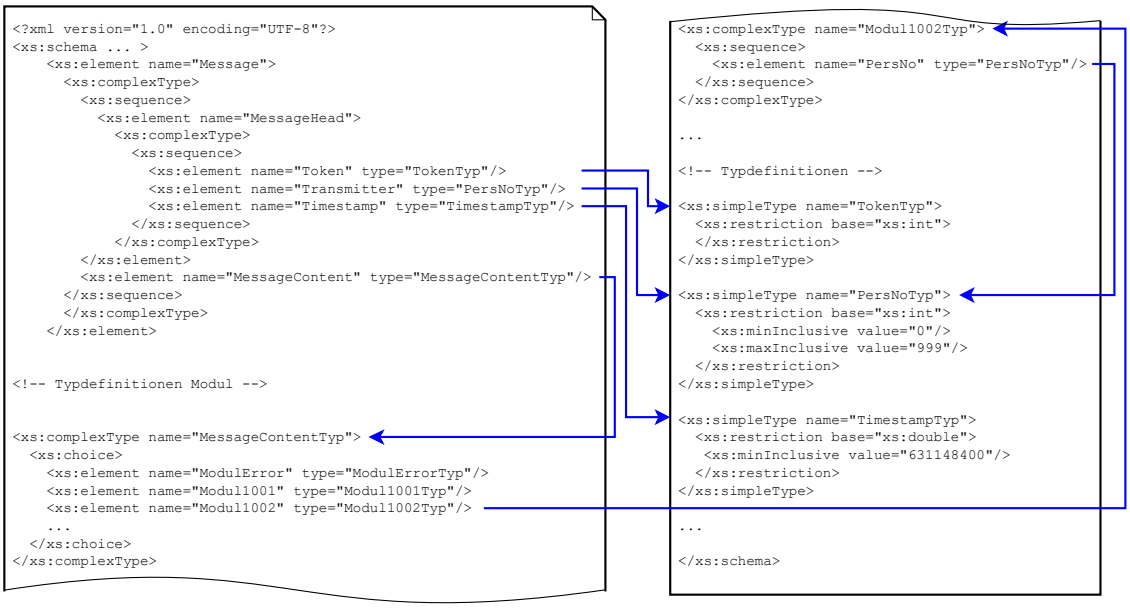

Abbildung 4.3: Auszug aus dem *XML*-Schema mit Augenmerk auf die Beziehungen zwischen Elementen und deren Typdefinitionen

Die Abbildung zeigt einen kleinen Ausschnitt aus dem verwendeten *XML*-Schema, welches im Anhang B.2 zu finden ist. Als erstes wird eine Nachricht definiert, nähere Erläuterungen dazu folgen im nächsten Kapitel. Die darin enthaltenen (Unter-)Elemente verweisen auf definierte Typen. Diese Beziehungen zwischen den Elementen und den Typdefinition werden mittels der blauen Pfeile aufgezeigt. Durch die Element-Typ-Beziehung können sehr gut Hierarchien abgebildet werden. Dabei befinden sich in der untersten Hierarchieebene elementare Datentypen. Zum Beispiel handelt es sich „PersoNoTyp“ um eine Ganzzahl, welche auf Werte aus dem Bereich 0...999 beschränkt wurde.

Die genannten Vorteile begründen die Entscheidung *XML* als Medium für die entste-

hende Kommunikation einzusetzen. Anzumerken ist allerdings die nachteilige Vergrößerung der zu übertragenen Daten gegenüber z. B. binärer Formate. Dies liegt zum einen am Textformat und zum anderen an der Beschreibung der Elemente mittels Tags. Dieser Nachteil bietet aber wiederum gleichzeitig einen Mehrwert bezüglich der Lese- und Debugbarkeit von Nachrichten.

4.1.4 Aufbau einer Nachricht

Jede Nachricht besteht aus zwei Teilen - dem Nachrichtenkopf (head) und dem Inhalt (content). Der Kopf einer jeden Nachricht ist gleich strukturiert und enthält den Absender (transmitter), ein Authentifizierungssymbol (token) sowie einen Absendezeitstempel (timestamp). Wohingegen der Inhalt einer jeden Nachrichten unterschiedlich ist. Die definierten Inhalte einer Nachricht werden Modul genannt. Pro Nachricht wird immer genau ein Modul versendet. Die Module klassifizieren sich in vier Bereiche.

- Modul 1000..1999: Anfrage zu Avero-Inhalten
- Modul 2000..2999: Anfrage zu Wrapper-Inhalten
- Modul 5000..5999: Antwort einer Anfrage zu Avero-Inhalten
- Modul 6000..6999: Antwort einer Anfrage zu Wrapper-Inhalten
- Modul Error: Antwort mit Fehlerinformationen

Die selbst definierte Klassifikation soll als grobe Einordnung der Nachrichtengattung dienen. Es ist allerdings nicht auszuschließen, das z. B. bei einer Anfrage nach Wrapper-Inhalten auch die Avero-Datenbank für Unterabfragen verwendet wird. Jedem Anfragemodul ist genau ein Antwortmodul zugeordnet. Durch die Addition der Anfragemodulnummer mit dem Summanden 4000 erhält man die Antwortmodulnummer. Dies hat den Vorteil, dass bei einer Anfrage durch die Clientapplikation, z. B. Modul 2012, diese eine bestimmte Reaktion vom Wrapper erwarten kann, in diesem Fall Modul 6012. Einzige Ausnahme von dieser Regelung ist das Verhalten bei einem Fehler. Dann erhält die Clientapplikation als Antwort kein Antwortmodul, sondern das Fehlermodul, welches allgemeingültig ist. Abbildung 4.4 zeigt das Anfragemodul 1002 sowie das dazugehörige Antwortmodul. Das Anfragemodul fragt hierbei die aktuellen Bestandsdaten des Urlaubskontos vom Mitarbeiter mit Personalnummer 815 an. Das Antwortmodul meldet 23 Urlaubstage als Ergebnis.

Eine *XML*-Nachricht beginnt immer mit zwei einleitenden Zeilen. In der ersten Zeile wird die *XML*-Version sowie die Art der Textkodierung angegeben. In der zweite Zeile beginnt die Nachricht unter Angabe zweier optionaler Attribute zum verwendeten Namensraum sowie gegen welches Schema geprüft werden soll. Beide Attribute wurden allerdings aus Platzgründen nur angedeutet. Ab der 3.Zeile beginnt der Nachrichtenkopf mit dem einleitenden Tag $< MessageHead >$. Dieser enthält wie erwähnt Zusatzinformationen zur Kommunikation. Das eigentliche Modul befindet sich jeweils zwischen den

Modul 1002 (Request)

```
<?xml version="1.0" encoding="UTF-8"?>
<Message ... >
    <MessageHead>
        <Token>12545</Token>
        <Transmitter>815</Transmitter>
        <Timestamp>631148444</Timestamp>
    </MessageHead>
    <MessageContent>
        <Modul1002>
            <PersNo>815</PersNo>
        </Modul1002>
    </MessageContent>
</Message>
```

Modul 5002 (Response)

```
<?xml version="1.0" encoding="UTF-8"?>
<Message ... >
    <MessageHead>
        <Token>0</Token>
        <Transmitter>815</Transmitter>
        <Timestamp>631148444</Timestamp>
    </MessageHead>
    <MessageContent>
        <Modul5002>
            <PersNo>815</PersNo>
            <Holiday>23</Holiday>
        </Modul5002>
    </MessageContent>
</Message>
```

Abbildung 4.4: Beispielhafte Kommunikation mittels *XML*-Nachrichten

$< MessageContent >$-Tags. Das Anfragemodul besitzt nur ein Element. Das Antwortmodul, welches der Wrapper antwortet, enthält im dargestellten Beispiel zwei Elemente. Den Abschluss bildet das schließende Tag $< /Message >$.

Alle Module sind in der Schnittstellen-Dokumentation im Anhang A.3 ausführlich beschrieben.

4.2 Nachrichtenverarbeitung

Zwischen dem Empfangen einer Anfrage und dem Senden der Antwortnachricht sind sieben thematisch voneinander getrennte Aufgaben vom Wrapper zu erfüllen.

1. Validierung
2. Deserialisierung
3. Injektionsprüfung
4. Berechtigungsprüfung
5. Informationsverarbeitung und -bereitstellung
6. Serialisierung
7. ggf. Fehlerbehandlung

Diese Aufgaben werden nachfolgend kurz erläutert und auf die Besonderheiten hingewiesen.

4.2.1 Validierung

Eine eingehende Nachricht muss auf deren Validität (lat. validus: gesund) geprüft werden. Diese Gültigkeitsprüfung bezieht sich allerdings nur auf die Nachricht als solche

und nicht auf deren Semantik. Die Prüfung erfolgt mit den in Kapitel 4.1.3 diskutierten *XML*-Schemata. Für alle Nachrichten wird in diesem Anwendungsfall auf nur ein Schema verwiesen, welches im Anhang B.2 zu finden ist. In diesem Schema sind die Strukturierung einer Nachricht, der Inhalt aller Module sowie die typbezogenen Beschränkungen definiert.

Für die Validierung wird nun zunächst die Schema-Datei eingelesen und als solche bereitgestellt. Nun kann die *XML*-Nachricht, welche als String oder Stream vorliegt, Zeile für Zeile gelesen und geprüft werden. Kommt es dabei zu einem Widerspruch, wird eine Ausnahme ausgelöst. Wertet man diese Ausnahme aus, so stehen detaillierte Informationen zur erkannten Unstimmigkeit bereit. Abbildung 4.5 zeigt eine solche Unstimmigkeit an einem einfachen Beispiel sowie die dazu generierten Hinweise zur Fehlersuche.

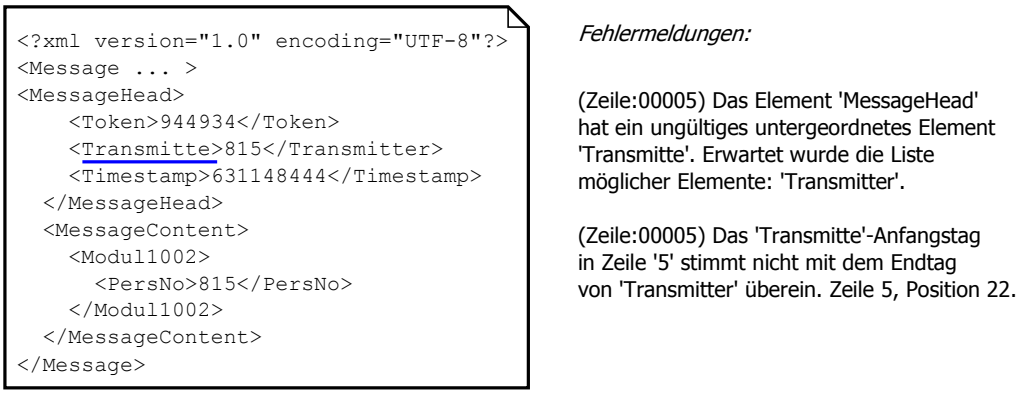

Abbildung 4.5: Fehlerhafte *XML*-Nachricht und die dazugehörigen Fehlermeldungen

Die Verarbeitung einer nicht erfolgreich validierten Nachricht ist nicht zielführend. Zum einen kann die nachfolgende Deserialisierung fehlschlagen oder es kommt in weiteren Prüf- oder Verarbeitungsschritten zu unerwünschten Nebenwirkungen. Deswegen kann die Validierung mit einem wohldefinierten Schema als starker Filter gegen fehlerhafte oder unzulässige Nachrichten verstanden und eingesetzt werden. Kommt es, wie es der Normalfall ist, während der Überprüfung zu keinem Fehler und die Nachricht wird als „valid“ eingestuft, so kann im nächsten Schritt die Deserialisierung durchgeführt werden.

4.2.2 (De-)Serialisierung

Unter Serialisierung versteht man die Umwandlung von Daten. Abgeleitet vom Wort „seriell“ (lat. serere: nacheinander) werden dabei Datenstrukturen in eine sequenzielle Form überführt. Im Gegensatz dazu erhält man bei der Deserialisierung aus einem Datenfluss wiederum strukturierte Daten. Am bereits bewährten Beispielmodul soll die Begrifflichkeit in Abbildung 4.6 noch einmal versinnbildlicht werden.

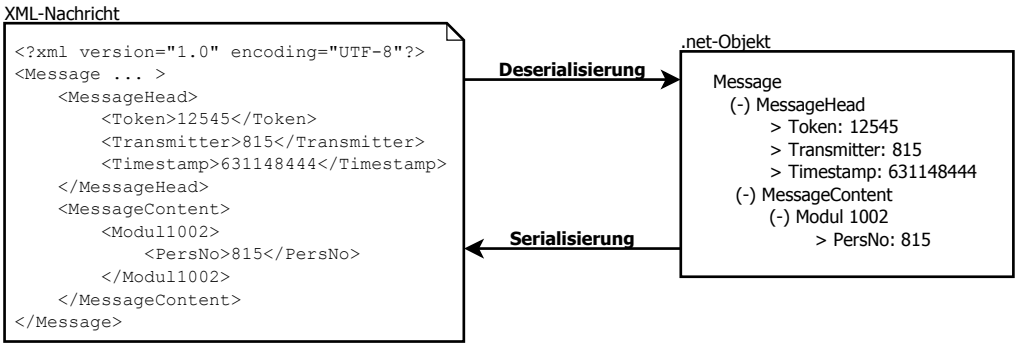

Abbildung 4.6: Unterschied zwischen de- und serialisieren

Bei dem verwendeten objektorientierten Programmierparadigma werden strukturierte Daten als Inhalt von Objekten aufgefasst. Kommt es nun zur Deserialisierung so werden die Objekte mit den sequenziellen Daten gefüllt. Um dies fehlerfrei zu gewährleisten müssen Objekte und Daten aufeinander abgestimmt werden. *XML* und die erwähnten Schemata bieten dafür eine prädestinierte Arbeitsumgebung. Es wäre durchaus möglich ohne die vorherige explizite Definition strukturierte Objekte zu erzeugen. Um allerdings Synergien in der objektorientierten Programmierung zu nutzen ist davon abzuraten.

Um eine erfolgreiche Deserialisierung einer validen Nachricht durchzuführen, gibt es allerdings eine Hürde. Man erlangt erst durch das Deserialisieren die Information um welche Daten es sich handelt bzw. welches Objekt damit gefüllt werden könnte. Die Objektklasse muss allerdings bereits vor dem Beginn der Umwandlung angegeben werden. Diese zyklische Kausalkette kann dahingehend gelöst werden, dass zu Beginn in eine anonymisierte Modulattrappe deserialisert wird. Diese kann wieder verworfen werden, nachdem das übermittelte Modul identifiziert wurde. Anschließend erfolgt eine zweite Deserialisierung in das nun bekannte Modul bzw. Objekt. Ein anderer Lösungsansatz ist, den Datenstrom sequentiell nach einer Modulnummer zu durchsuchen. Dies birgt allerdings Risiken und benötigt eine zusätzliche Implementierung eines Suchalgorithmus. Bei der verwendeten zweifachen Deserialisierung kann auf eine bestehende Funktionalität ohne Mehraufwand zurückgegriffen werden.

Mit der Klassenbibliothek „System.XML.Serialisation“ liefert das verwendete .NET-Framework geeignete Werkzeuge für die geplanten Tätigkeiten. Wie Abbildung 4.7 zeigt, kann mit wenigen Kodezeilen eine Methode zum Deserialisierung umgesetzt werden. Eine Methode zum Serialisieren ist ähnlich aufgebaut.

Um in diesem Fall nicht für jedes Anfragemodul (Request) eine eigene Methode zu definieren, sollte auf die Vererbungslehre der objektorientierten Programmierung zurückgegriffen werden. Diese ermöglicht im konkreten Anwendungsfall eine dynamische Angabe in welche Objektklasse deserialisiert werden soll. Das dazu zugrunde liegende Klassendiagramm wird in Kapitel 5.2 näher erläutert.

```
public bool DoDeserialize(System.Type pTypObj, ref MyMessages.RequestMessage pObj, string pRequest)
{
    //Deserialisier erstellen
    System.Xml.Serialization.XmlSerializer deserializer = new System.Xml.Serialization.XmlSerializer(pTypObj);

    //String in Stream wandeln
    byte[] readByte = System.Text.Encoding.UTF8.GetBytes(pRequest);

    //Einleser erstellen
    System.IO.MemoryStream reader = new System.IO.MemoryStream(readByte);

    try      //Versuche in Objekt zu deserialisieren
    {
        pObj = (MyMessages.RequestMessage)deserializer.Deserialize(reader);
        return true;
    }
    catch    //Versuch fehlgeschlagen
    {
        return false;
    }
    finally
    {
        reader.Close();
        deserializer = null;
    }
}
```

Abbildung 4.7: Beispielhafte Methode zur Deserialisierung einer *XML*-Nachricht in das entsprechende Objekt mit Hervorhebung der verwendeten Klassenhierarchie

4.2.3 Injektionsprüfung

Durch den Einsatz des Wrappers als Vermittler zwischen Clientapplikation und Datenbanksystem ergeben sich einige Vorteile. Einer davon war die Zentralisierung der Datenbankabfragen. Somit besitzt kein Client eine Autorisierung um direkt auf die Datenbank zuzugreifen. Lediglich der Wrapper besitzt diese Anmeldedaten. Durch diese Struktur wird die Sicherheit erhöht. Per definitionem ist aber keine absolute, sondern lediglich eine statistische Sicherheit möglich.[18] Um Gefahren aufzudecken, soll die resultierende Datenbankabfrage für das bereits vorgestellte Modul 1002 konkret betrachtet werden.

```
SELECT PNo,Info2 FROM Avero.dbo.InfoValues
WHERE PersNo=815 AND TM=0
```

Bevor diese Anfrage vom Wrapper gestellt werden kann, muss dieser sicherstellen, dass der Absender der Anfrage berechtigt ist, diese Information abzufragen. Diese und weitere Berechtigungen werden im nachfolgenden Kapitel erläutert. Die Fragestellung einer Berechtigung, gültigen Validierung oder möglichen Deserialisierung soll im Augenblick unberücksichtigt bleiben. Aktuell soll erörtert werden, ob und wie man diese Abfrage beeinflussen kann, um beispielsweise erweiterten Zugriff auf die Datenbank zu erhalten.

Um zu zeigen wie dies möglich ist, soll erläutert werden, wie die Clientapplikation oder ein Angreifer diese Anfrage beeinflussen und ggf. verändern kann. Der *SQL*-Query besteht aus einem gleichbleibenden und einem dynamischen Anteil. Der dynamische An-

[18] International Organization for Standardization 2013

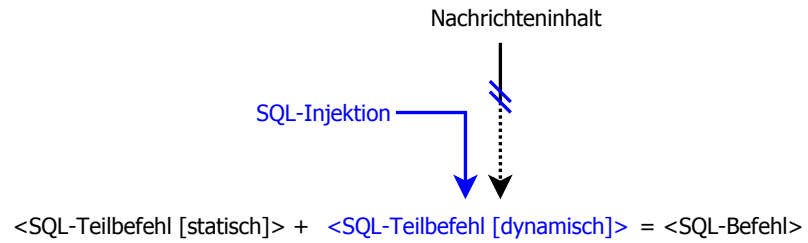

Abbildung 4.8: Prinzip einer *SQL*-Injektion durch die Beeinflussung dynamischer *SQL*-Teilbefehle

teil wird von dem Wrapper aus dem *XML*-Tag „PersNo“ des empfangenen Moduls 1002 deserialisiert. Verändert man den Inhalt dieses *XML*-Elements, so verändert man auch direkt die Datenbankanfrage. Mit dieser Erkenntnis liegt der Schluss nahe, genau diesen dynamischen Anteil sich zunutze zu machen. Injiziert man nun diesen Anteil mit schädlichem Inhalt, so spricht man von einer *SQL*-Injektion, deren Prinzip in Abbildung 4.8 dargestellt ist.

Um die Gefahr zu verdeutlichen, könnte man eine eigene Abfrage, wie die folgende, zur Datenbank absetzen. Damit würde man die angefragten Informationen nicht nur zu einem Mitarbeiter, sondern zu allen Mitarbeitern erhalten.

```
SELECT PNo,Info2 FROM Avero.dbo.InfoValues
WHERE TM=0
```

Ist man sich bewusst, dass die Datenbank auch gleichzeitig mehrere Anfragen verarbeiten kann, so liegt der Lösungsansatz für einen potenziellen Angriff nah. Fügt man beide Anfragen zusammen, wie nachstehend getan, so werden beide Anfragen von der Datenbank abgearbeitet. Der ursprüngliche dynamische Anteil, ergo der Inhalt des *XML*-Elements, wurde einfach um die zusätzliche Abfrage erweitert und mit einem Semikolon separiert.

```
SELECT PNo,Info2 FROM Avero.dbo.InfoValues
WHERE PersNo=815 AND TM=0; SELECT PNo,Info2
FROM Avero.dbo.InfoValues WHERE TM=0
```

Sicherlich gibt es noch das Hindernis, die zusätzlich gewonnenen Informationen erfolgreich zu verarbeiten. Aber bedenkt man, dass es auch Abfragen gibt, welche den Datenbestand verändern oder löschen ohne eine weitere Informationsverarbeitung zu benötigen, ist die vorhandene Sicherheit gegen einen versierten Angreifer durch die Zentralisierung verschwindend gering.

Eine Überprüfung der Inhalte der *XML*-Elemente ist gerade im Zusammenhang mit Datenbanken zwingend erforderlich. Zum Beispiel stellt die Überprüfung mittels einer „Blacklist“ eine Erhöhung der Sicherheit dar. In der „Blacklist“ sind unzulässige Wörter

oder Zeichen enthalten, wie „;“ oder „delete“. Es ist zu beachten, dass vor der Prüfung mittels Blacklist das Format des Inhaltes angepasst wird. Damit wird vermieden, dass ein „delEte“ nicht erkannt wird.

Durch die Injektionsüberprüfung konnte ein probates Mittel gegen unzulässige Zugriffe auf die Datenbank gefunden werden. Handelt sich nun um eine valide und injektionsfreie Nachricht, welche erfolgreich deserialisiert werden konnte, so ist anschließend die Semantik und die Autorisierung dieser zu prüfen.

4.2.4 Berechtigungsprüfung

Konnte die Nachricht bis zur Berechtigungsprüfung vordringen, so stimmt die äußere Form dieser. Nun ist es an der Zeit die eigentliche Sinnhaftigkeit sowie das Autorisierungsrecht der Anfrage zu prüfen. Unter Sinn ist in diesem Zusammenhang die Korrespondenz zum bestehenden Datenbestand zu verstehen. Betrachten wir dazu erneut das Modul 1002. Eine Anfrage, welche dieses Modul enthält, muss folgende Prüfungen positiv durchlaufen.

- gültiges Autorisierungssymbol (= validToken(Token))
- gültiger Absender (= validEmployee(Transmitter))
- gültiger Mitarbeiter (= validEmployee(PersNo))
- Autorisierung für Informationen (= validMe(Transmitter,PersNo)
  oder validSuperior(Transmitter,PersNo))

Wenn der Nutzer die Clientapplikation startet, muss er sich mit seiner Personalnummer sowie dem gültigen Passwort anmelden. Stimmt das Passwort, so erhält die Clientapplikation ein gültiges Authentifizierungssymbol vom Vermittler. Fortan erfolgt die Authentifizierung nicht mehr mit dem Passwort, sondern über das 6-stelligen Symbol. Ist der übertragene Token allerdings ungültig, wird die Anfrage abgewiesen. Weiterhin muss der Absender sowie der angefragte Mitarbeiter ein gültiger Nutzer sein. Abschließend wird geprüft, ob der Anfrager berechtigt ist, die angeforderten Informationen zu erhalten. In diesem Kontext ist dies nur gegeben, wenn der Anfrager der Mitarbeiter selbst oder sein Vorgesetzter ist. Schlägt eine dieser Prüfungen fehl, so wird die Nachricht nicht verarbeitet. Der Client erhält in diesem Fall als Antwort das Modul Error.

Für die verschiedenen Module gilt es unterschiedliche Berechtigungen zu prüfen. Allerdings gibt es einige Prüfungen, die bei sehr vielen Anfragemodulen durchgeführt werden, z.B. nach einem gültigen Autorisierungssymbol oder Mitarbeiter. Eine vollständige Tabelle aller Prüfungen inklusive der Zuordnung zu den einzelnen Modulen befindet sich im Anhang A.2.

4.2.5 Informationsverarbeitung und -bereitstellung

Sind alle Kontrollen im Vorfeld erfolgreich durchlaufen, so gilt es nun die Anfragen zu bearbeiten und ein entsprechendes Antwortmodul zu generieren. Hierfür löst ein Anfragemodul eine oder mehrere Datenbankabfragen aus. Dazu wird mittels der übergebenen Informationen der eingegangenen Nachricht ein *SQL*-Query gebildet und an die entsprechende Datenbank gesendet. Diese antwortet mit einem *SQL*-Result, welcher anschließend ausgewertet wird. Entweder sind schon alle Informationen für das Antwortmodul enthalten oder es erfolgen weitere Datenbankabfragen. Im Beispielmodul 1002 wird mittels einer Datenbankabfrage die benötigte Information zum Resturlaub eines Mitarbeiters bereitgestellt.

Anschließend wird ein neues Antwortobjekt erzeugt und die dazugehörigen Kopfdaten generiert. Das entsprechende Antwortmodul (Anfragemodul + 4000) bildet den eigentlichen Inhalt der Nachricht. Im genannten Beispiel ist dies Modul 5002 mit der ermittelten Anzahl des Resturlaubs.

Anschließend erfolgt die in Kapitel 4.2.2 beschriebene Serialisierung. Dabei wird aus dem erzeugten Antwortobjekt eine *XML*-Antwortnachricht generiert. Diese Nachricht wird abschließend an die Clientapplikation gesendet. Dort wird sie wiederum verarbeitet, ggf. aufbereitet und visualisiert.

4.2.6 Fehlerbehandlung

Die Qualität der Clientapplikation legt den Grundstein für eine geringe Fehlerquote beim Abarbeiten der Anfragen durch den Wrapper. Sendet der Client nur qualifizierte Nachrichten im Sinne der äußeren Struktur, einem schadfreien Inhalt, der Sinnhaftigkeit sowie ausreichender Berechtigung, würde sich eine Fehlerbehandlung fast erübrigen. Aber auch für eine unqualifizierte Nachricht muss der Vermittler eine passende Antwort parat haben. Diese sollte dem Anfrager auf den begangenen Fehler hinweisen und ihm helfen seine Anfrage ggf. anders zu formulieren. Keinesfalls sollte eine fehlerhafte Anfrage zum Absturz des Wrappers führen, da dieser schließlich auch für Anfragen anderer Nutzer zur Verfügung stehen muss.

Kommt es in den genannten 6 bzw. 7 Einzelschritten zu einem Fehler, so muss dieser erfasst, zwischengespeichert und daraus eine Antwort mit einem Fehlermodul generiert werden. Die Erfassung erfolgt in dem jeweiligen Schritt, wo z. B. eine Prüfung fehlschlägt. Dazu meldet die Instanz den Fehler an ein übergeordnetes Fehlermanagement, dem sogenannten „Log-Center“. Neben Fehlern können hier auch beliebige Meldungen zwischengespeichert werden. Dies unterstützt ggf. eine Fehlersuche. Bei Beendigung des fehlerhaften Einzelschritts wird ein Fehlerflag gesetzt. Dieses Fehlerflag bewirkt, dass die noch folgenden Einzelschritte übersprungen werden, denn die Bearbeitung

dieser ist wenig zielführend. Führt beispielhaft die Validierung gegen das *XML*-Schema zu einer Unstimmigkeit, so ist in den meisten Fällen weder eine Deserialisierung möglich noch eine Informationsverarbeitung sinnvoll.

Ein Fehlermodul kann aus mehreren Meldungen bestehen. Die Meldungen sind chronologisch geordnet. Zu jeder Meldung gehört ein Fehlerort bzw. die meldende Instanz sowie ein Meldetext zur genaueren Fehlerqualifizierung. Auch das Fehlermodul wird abschließend in eine *XML*-Nachricht serialisiert und an den anfragenden Client gesendet. Das Beispiel in Abbildung 4.9 zeigt die Reaktion auf eine fehlerhafte Anfrage mit Modul 1002.

```
<?xml version="1.0" encoding="utf-8"?>
<Message ... >
  <MessageHead>
    <Token>0</Token>
    <Transmitter>0</Transmitter>
    <Timestamp>631148444</Timestamp>
  </MessageHead>
  <MessageContent>
    <ModulError>
      <Error>
        <ErrorLocation>XML-Validator</ErrorLocation>
        <ErrorMessage>Nachricht hat Fehler: (Zeile:00005) Das Element...</ErrorMessage>
      </Error>
      <Error>
        <ErrorLocation>XML-Validator</ErrorLocation>
        <ErrorMessage>Nachricht hat Fehler: (Zeile:00005) Das 'Transmitte'-Anfangstag...</ErrorMessage>
      </Error>
      <Error>
        <ErrorLocation>XML-Validator</ErrorLocation>
        <ErrorMessage>Fehler in der Konfiguration!</ErrorMessage>
      </Error>
      <Error>
        <ErrorLocation>Nachrichten-Verwaltung</ErrorLocation>
        <ErrorMessage>Nachricht kann nicht validiert werden.</ErrorMessage>
      </Error>
    </ModulError>
  </MessageContent>
</Message>
```

Abbildung 4.9: Beispielhafte *XML*-Nachricht mit Fehlermodul

5 Softwareentwurf

Dieses Kapitel befasst sich mit dem Entwurf der eigentlichen Softwarestruktur. Dazu wird das Handling (dt. Abwicklung) von parallel eintreffenden Nachrichten betrachtet. Anschließend wird die Klassenstruktur anhand der Modulverarbeitung erörtert. Das diesbezügliche Klassendiagramm zeigt dabei die verschiedenen Beziehungen unter den Klassen.

5.1 Synchronisierung

Der Vermittler muss Anfragen mehrerer verschiedener Client(-applikationen) gleichzeitig annehmen können. Ein System, welches diese Fähigkeit besitzt, wird als Multitasking-System bezeichnet. Dabei verlaufen die zu bearbeitenden Aufgaben allerdings nur quasi parallel. Kommt es hingegen zu einer echten Nebenläufigkeit in der Verarbeitung, spricht man von einem Multiprozessorsystem. Dabei ist die verwendete Hardware i. d. R. maßgebend.[19] Der Wrapper wird aus diesem Grund nur als Multitaskingsystem ausgelegt. Doch auch in diesem Anwendungsfall kann es bereits durch eine unzureichende Synchronisation zwischen den Programmteilen zu einer fehlerhaften Bearbeitung oder sogar zu Deadlocks kommen.

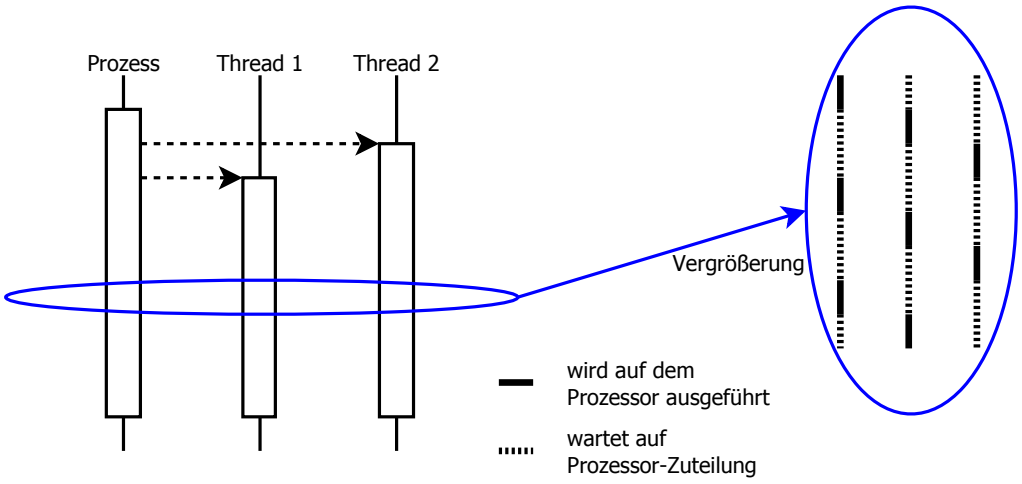

Abbildung 5.1: Quasi Nebenläufigkeit eines Prozesses und den dazugehörigen Threads mit Round-Robin-Scheduling

Um die Technologie zu nutzen, wird für jede eingehende Nachricht ein Thread (dt. Faden) erzeugt. Ein Thread kann als selbstständiger, ausgelagerter Programmabschnitt verstanden werden. Diese Programmfäden gehören weiterhin zum erzeugenden Prozess. Durch schnelles Zeitmultiplexen wird nun zwischen dem Hauptprozess und den

[19] Vgl. Spinczyk 2010, S.10

erzeugten Threads umgeschaltet. Das Umschalten geschieht in sehr kurzen Zeitabständen, z. B. ca. 10 ms[20], sodass die erwähnte quasi Nebenläufigkeit erzeugt wird, wie Abbildung 5.1 zeigt.

Durch diese Art der Parallelität können nun Programmabschnitt eigenständig Ihre Arbeit verrichten. Allerdings gibt es keine Schutzmechanismen, die unzulässige Zugriffe untereinander verhindern. Alle Threads haben einen gemeinsamen Speicherbereich und Kontext. Ebenfalls ist der Zugriff auf Ressourcen und deren Veränderung von allen Threads jederzeit möglich. Dies führt zu sogenannten Race Conditions (dt. kritischer Wettlauf) um Ressourcen. Solche Wettlaufsituationen und ihre Folgen, wie in Abbildung 5.2 dargestellt, sind für einen fehlerfreien Betrieb zu vermeiden.

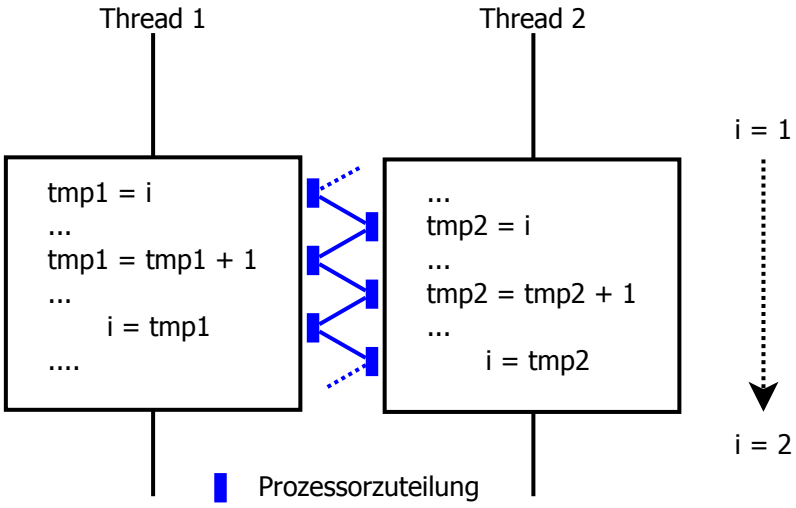

Abbildung 5.2: Wettlaufsituation um die Variable i führt zu falschem Ergebnis

Ein geordneter Zugriff auf Ressourcen ist daher zwingend erforderlich. Wird eine Ressource gerade durch einen Thread verwendet, so muss sie für einen anderen gesperrt sein. Die anderen Threads warten nun auf die exklusive Ressource. Ein solches Warten wird in der Software-Entwicklung auch als wechselseitiger Ausschluss bezeichnet. Es tritt eine Synchronisation ein. Für dessen Realisierung gibt es verschiedene Mittel. Mittels *FIFO*-Queue (dt. Warteschlange) kann die Funktionalität einer Warteschlange nachgebildet werden. Ein einfacheres Mittel, welches hier zur Anwendung kommen soll, ist eine Sperre. Dieses sogenannte Lock (dt. Türschloss) arbeitet wie eine Schranke, welche den Eintritt in einen kritischen Prozess-/Kodeabschnitt versperren kann. Bei einem Lock kann immer nur ein Thread/Prozess den kritischen Abschnitt betreten. Um die Sperren geeignet zu verteilen, muss man sich die exklusiven Ressourcen sowie die kritischen Abschnitte vergegenwärtigen. Die Tabelle 5.1 gibt Aufschluss über die exklusiven Ressourcen sowie die eingesetzten Sperren.

Dabei ist zu beachten, dass nicht nur der Ressourcenzugriff limitiert wird, sondern dass auch keine Race Condition in Bezug auf den Inhalt der Nachrichten eintritt. Dies wä-

[20] Vgl. Eisenhardt 2007, S.102

Bereich	Schritt	Sperre	exklusive Ressource
I	Validierung	lockerXMLValidator	XML-Validator
II	Deserialisierung	lockerXMLSerialsator	XML-Serialsator
III	Injektionsprüfung	lockerSQLInjectionChecker	SQL-Injection-Checker
IV	Berechtigungsprüfung, Informationsverarbeitung	lockerDatenbase	Datenbankzugriff, -inhalt
V	Serialisierung	lockerXMLSerialsator	XML-Serialsator
VI	Fehlerbehandlung	lockerXMLSerialsator	XML-Serialsator

Tabelle 5.1: Übersicht über die einzelnen Schritte mit den eingesetzten Sperren für die exklusiven Ressourcen

re der Fall, wenn z. B. zwei Nachrichten den gleichen Inhalt der Datenbank verändern möchten, analog dem Beispiel aus Abbildung 5.2. Deswegen muss die Grundregel für die Definition von kritischen Abschnitten: „Fasse einen kritischen Abschnitt so kurz wie möglich“, um den Nachsatz: „...und so lang wie nötig.“ erweitert werden.[21] Um im betrachteten Anwendungsfall die erwähnte Wettlaufsituation zu vermeiden, müssen die Bereiche relativ groß gewählt werden, siehe Bereich IV. Da es sich herbei allerdings nicht um ein Multiprozessorsystem handelt, ist der damit einhergehende Leistungsverlust zu vernachlässigen. Die Sperren werden zentral in der Klasse „MessageManager“ implementiert, welche die Nachrichtenverarbeitung laut den in Kapitel 4.2 beschriebenen Einzelschritten realisiert.

5.2 Klassenstruktur und -diagramm

Der methodische Aufbau der Klassenstruktur soll am Beispiel der Nachrichtenklasse erläutert werden. Es wurde dabei Wert auf die beiden Konzepte Vererbung und Polymorphismus (dt. Vielgestaltigkeit) der objektorientierten Programmierung gelegt. Im Gegensatzsatz zur prozeduralen Herangehensweise bringt dieser Programmstil viele Synergieeffekte mit sich. Objekte und Daten werden dabei nicht mehr getrennt voneinander betrachtet, sondern verschmelzen zu kleinen logischen Blöcken. Diese logischen Blöcke, auch Klassen genannt bzw. Instanzen davon, werden in Beziehungen zueinander gestellt. Diese Beziehungen können anhand eines Klassendiagramms dargestellt werden. Es werden darin nicht nur Klassen sowie deren Methoden und Attribute benannt, sondern auch Vererbungshierarchien aufgezeigt. Das Klassendiagramm in Abbildung 5.3 zeigt die Struktur um die Klasse „ResponseMessage“. Die Aufgabe dieser Klasse(-nstruktur) ist die Umwandlung der Informationen aus den Datenbanken in Module, welche später in *XML*-Nachrichten serialisiert werden.

Im dargestellten Diagramm sind Abhängigkeiten wie Vererbung (Verbindungslinie mit

[21] Vgl. Quade 2011, S.206 ff.

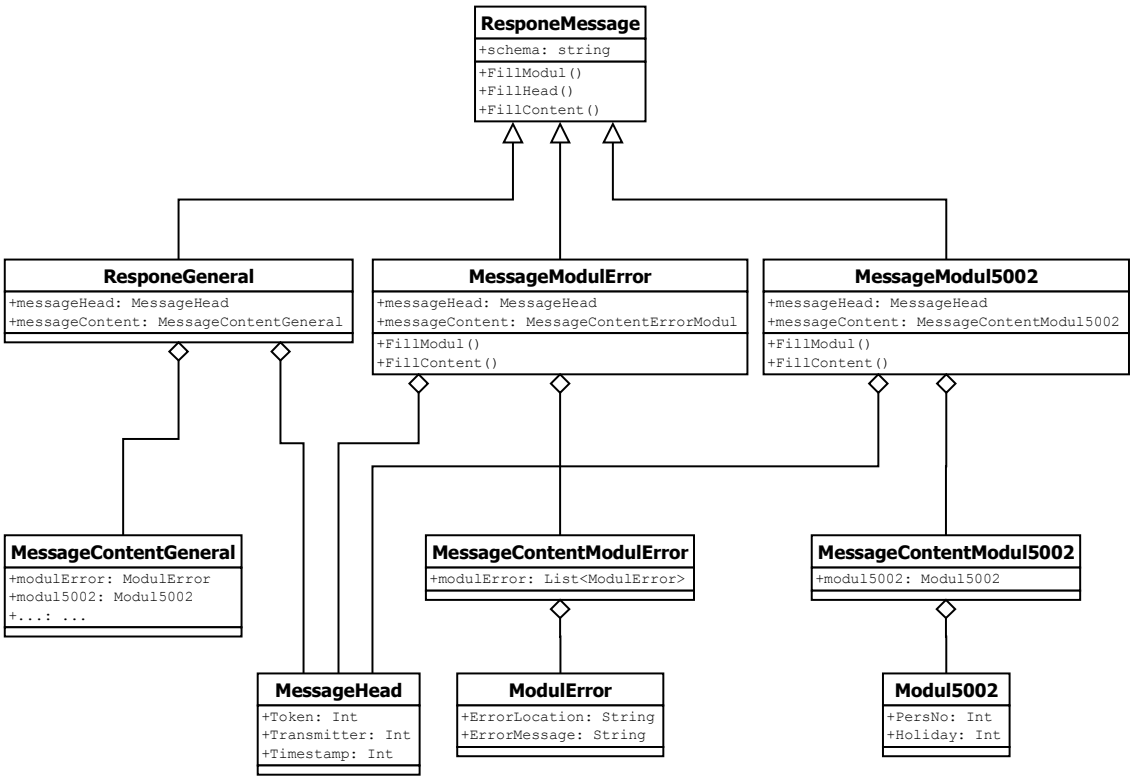

Abbildung 5.3: Auszug aus dem Klassendiagramm um die Klasse ResponeMessage

Pfeil) und Assoziationen (Verbindungslinie mit Raute) sichtbar. Die starke Klassifizierung in (Unter-)Klassen liegt nicht allein in der damit einhergehenden Vereinfachung begründet. Die Ursache liegt vielmehr in der Struktur des XML-Schemas, welche stark gegliedert ist. Der angewendete Polymorphismus ist in der Klasse „ResponeMessage“ sehr gut sichtbar. Durch die Vererbung können die spezielleren (Kind-)Klassen vorteilhaft über die generelle (Eltern-)Klasse angesprochen werden. Wobei die generelle Klasse z. B. abstrakt ist oder deren Methoden nur „virtual“ angedeutet sind. In diesem Zusammenhang spricht man auch von Generalisierung. Neben den bereits erwähnten Konzepten ist die Kapselung, z. B. von privaten Daten, die dritte Säule der objektorientierten Programmierung. Dabei können klassen- bzw. instanzinterne Daten vor dem Zugriff von außen geschützt oder beschränkt werden. Sie soll allerdings hier nur eine untergeordnete Rolle spielen.

6 Implemetierung

6.1 Datenübertragung und Verschlüsselung

Das .NET-Framework bietet im „System.Net“-Namespace Klassen für eine Server-Client-Kommunikation auf Basis des „Hypertext Transfer Protocol“ an. Für den Server wurde die Klasse „HttpListener“ bzw. für die Clientapplikation „HttpWebRequest“ und „HttpWebResponse“ verwendet. Weiterhin wurde ein synchrones Kommunikationsprinzip mit der GetContext()-Methode verwendet. Dabei wird zeitgleich immer nur eine Anfrage empfangen. Ein bespielhafter Quellkode für eine Kommunikationsabwicklung mittels einem HTTPS-Server und HTTPS-Client ist im Anhang B.3 enthalten.

Dabei ist zu beachten, dass vor dem Start der eigentlichen Kommunikation, der Server gestartet und eingerichtet sein muss. Erreicht die Server-Applikation die GetContext()-Methode, so wartet diese auf Anfragen von Clients. Sinngemäß erwartet bzw. lauscht (engl. listen) der Server nach eingehenden Nachrichten. Trifft eine neue Anfrage ein, so erfolgt die Weiterverarbeitung durch den Server. Dieses Warten auf die andere Applikation ist der Grundgedanke einer synchronen Kommunikation. Allerdings soll nicht die gesamte Server-Applikation warten. Vielmehr ist dies nur für das entsprechende Teilsystem beabsichtigt, sodass z. B. die Applikation weiterhin auf Benutzereingaben reagiert. Deswegen wird die Kommunikation in Threads ausgelagert, wie bereits im Kapitel 5.1 erwähnt.

Bis dahin erfolgt der Datenaustausch zwischen Client und Server unverschlüsselt. Mittels „Hypertext Transfer Protocol Secure“ kann dies geändert werden, wie in Kapitel 4.1.2 erläutert. Soll *HTTPS* aktiviert werden sind folgenden Schritte notwendig.

1. erstellen/anfordern eines Zertifikates
2. importieren dieses Zertifikats in die Zertifikatsverwaltung des Servers
3. binden des Zertifikats an die Anschlusskonfiguration
4. Implementierung des neuen Protokollaufrufs und Ports
5. Kontrolle der Verschlüsselung

Es gibt mehrere Möglichkeiten ein Zertifikat zu erlangen. Zum Beispiel kann dies durch eine Zertifizierungsstelle ausgestellt oder selbst erstellt werden. Ein selbst ausgestelltes Zertifikat ist für den Anwendungsfall ausreichend. Mittels des Konsolenwerkzeuges „Makecert“[22] der Microsoft Windows *SDK* und dem nachfolgenden Aufruf kann ein solches erzeugt werden. In dem Aufruf werden u. a. Informationen zum Zertifikatsname,

[22] Microsoft - Makecert

der Gültigkeitszeitraum, die Schlüsselverwendung und der CryptoAPI-Anbieter festgelegt.

```
makecert -r -pe -n "CN=FHRbooking" -b 01/01/2000
-e 01/01/2036 -eku 1.3.6.1.5.5.7.3.1 -ss my
-sr localMachine -sky exchange
-sp "Microsoft RSA SChannel Cryptographic Provider"
-sy 12
```

Das Zertifikat muss anschließend noch in den zentralen Zertifikatsspeicher des Servers unter „Lokaler Computer/vertrauenswürdige Stammzertifizierungsstellen“ importiert werden. Danach müssen Server-Applikation, Zertifikat und Netzwerkanschluss (IP-Adresse und/oder Portnummer) miteinander verbunden werden. Diese Netzwerkeinstellung wird mit dem Konsolenwerkzeug „netsh“ der Microsoft Windows NT-Linie und dem nachfolgenden Aufruf administriert. Der Aufruf enthält das verwendete Port 8443 („ipport“), den Fingerabdruck des erstellten Zertifikats („certhash“) sowie den “Globally Unique Identifier“ der Anwendung („appid“).

```
netsh http add sslcert ipport=0.0.0.0:8443
certhash=c303e55ac6d1ffc140d577a3f874f2df8e6ed698
appid="{413dfc09-99ab-4b94-9425-1c5507363b92}"
```

Sowohl bei der Client- als auch der Serveranwendung müssen die Adressaufrufe auf „https://[IPADRESSE]:[SSLPORT]“ umgestellt werden. Durch die Angabe *HTTPS* wird beiden Instanzen signalisiert, dass *SSL/TLS*-Handshake zu verwenden. Für die verschlüsselte Verbindung wird ein zweites Port verwendet. Das Standardport für *HTTPS* ist 443. Um Adresskonflikte im Vorfeld zu vermeiden, wurde das Port 8443 gewählt. Die Serversoftware wurde so konfiguriert, dass beim parameterlosen Aufruf nur das *SSL*-Port aktiv ist und somit ausschließlich eine sichere Verbindung möglich ist.

6.2 Datenbankzugriff

Der Wrapper soll als Vermittler zwischen der Clientapplikation und der bestehenden Datenbasis „Avero“ fungieren. Die erste genannte Teilstrecke davon wurde im vorangegangen Kapitel erläutert. Nun soll der zweite Abschnitt - die Verbindung zum Datenbankmanagementsystem (*DBMS*) - betrachtet werden. Dazu wird zum einen der Verbindungsaufbau näher beleuchtet und zum anderen soll beispielhaft ein verwendeter *SQL*-Befehl erläutert werden.

6.2.1 Zugriff

Eine Datenbank besteht aus zwei Teilen - dem Datenbankmanagementsystem und den eigentlichen Daten. Das *DBMS* ist u. a. für die Verwaltung, die Sicherheit und den Zugriff auf die Daten zuständig. Der Zugriff auf das Datenbanksystem erfolgt i. d. R. über eine Datenbankschnittstelle. Diese Schnittstelle stellt eine *API* für die Implementierung einer Datenbank in einer Anwendung bereit. Abbildung 6.1 stellt diese Zusammenhänge dar. Das betrachtete Datenbanksystem arbeitet mit dem Managementsystem „Microsoft SQL Server“. Der Zugriff darauf erfolgt mit der Datenbankschnittstelle „ADO.net“.

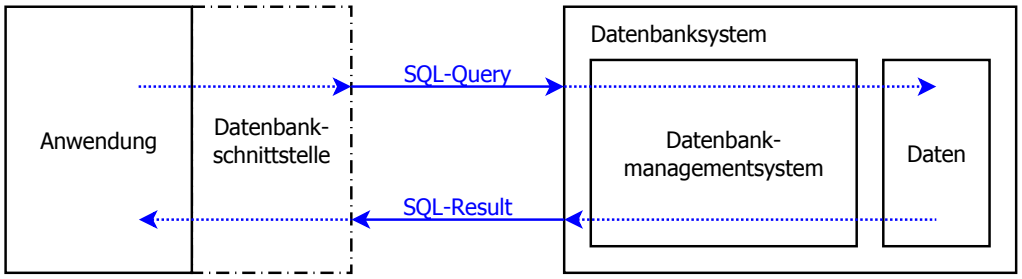

Abbildung 6.1: Aufbau des Datenbanksystems sowie dessen Interaktion mit der Anwendung über die Datenbankschnittstelle

Bevor der *SQL*-Query zur Datenbank gesendet werden kann, erfolgt eine Authentifizierung gegenüber dem Datenbanksystem mittels Benutzername und Passwort. Anschließend wird die Abfrage i. d. R. aus einem statischen und einem dynamischen Teil zusammengesetzt und gesendet. Das *SQL*-Result erreicht den Wrapper, je nach Abfrageumfang, in 1 bis 4 Sekunden. Das Ergebnis wird anschließend verarbeitet bzw. für die Weiterverarbeitung mittels Convert-, Parse- oder Castfunktionen aufbereitet.[23] Dabei ist zu beachten, dass teilweise auch „NULL“ Elemente enthalten sein können.

6.2.2 Beispielabfrage

Die verwendete relationale Datenbanksprache *SQL* wird i. A. in vier Sprachräume unterschieden: Datenverarbeitungs- (*DML*), Datenbeschreibungs- (*DDL*), Datenabfrage- (*DRL*) und Datensicherheitssprache (*DSL*). Oft ist es der Fall, dass Daten tabellenübergreifend abgefragt werden müssen. Für größere Datenmengen, welche zusammengeführt werden sollen, eignet sich das in Kapitel 3.2.2 vorgestellte „Data-Warehouse“. Für die effiziente Abfrage bzw. Verarbeitung von kleineren Datenmengen über mehrere Tabellen hinweg wird der *DRL*-Befehl JOIN (dt. vereinen) verwendet. Dabei wird zwischen (INNER) JOIN und OUTER JOIN unterschieden. Das Unterscheidungsmerkmal hierbei ist, ob Einträge, welche die Vergleichsbedingungen (ON) nicht erfüllen, dennoch erhalten bleiben oder nicht. Bei (INNER) JOIN werden nur gemeinsame Datensätze berücksichtigt. Diese Problematik trat häufig in der Arbeitsaufgabe auf und soll deswegen anhand des folgenden *SQL*-Querys erläutert werden.

[23] Vgl. High Flying

```
SELECT t2.Employee, t2.FavoritJob, t1.OperationDescription,
       t1.Customer, t1.Project, t1.ArticleNo, t1.Description
        FROM wrapper.dbo.Jobs t1
                JOIN  wrapper.dbo.FavoritJobs  t2
                ON  t1.job=t2.FavoritJob
        WHERE t2.Employee=1234
        ORDER BY FavoritJob ASC
```

Diese Abfrage ist im Modul 2006 „Anzeige favorisierte Jobs“ integriert und vereinigt Informationen aus den Tabellen „Wrapper.dbo.Jobs“ (= t1) und „Wrapper.dbo.FavoritJobs“ (= t2). Die Schnittmenge beider Tabellen ist das Attribut „Jobs“, welcher als Fremdschlüssel in der zweiten Tabelle eingesetzt wird. Daher wird „Jobs“ auch als Vergleichskriterium herangezogen. Für die Ausgabe werden nur Einträge berücksichtigt, die in beiden Tabellen vertreten sind - bezogen auf dieses Attribut. Aufgrund des Fremdschlüssels als Vergleichsattribut beinhaltet das Ergebnis dieser Abfrage alle Elemente der Tabelle „Wrapper.dbo.FavoritJobs“, allerdings nur ein Teil der Einträge der Tabelle „Wrapper.dbo.Jobs“. Durch die angehängte WHERE-Klausel kann das Resultat weiter eingeschränkt und durch die ORDER-BY-Angabe sortiert werden. In diesem Fall werden nur Einträge des Mitarbeiters mit der Personalnummer „1234“ ausgewählt und diese anschließend aufsteigend sortiert.

7 Integration in das bestehende System

7.1 Einbindung

Die Wrapper-Applikation soll auf dem gleichen Server arbeiten wie das bereits bestehende „Avero“. Dies ist nicht zwingend notwendig, vereinfacht allerdings die Administration und minimiert den Aufwand in Bezug auf Hardware-Technik und Software-Lizenzen. Nach Fertigstellung der Applikation wird diese auf den Server kopiert, mit dem Managementprogramm des Datenbanksystems die neue Datenbank erstellt, konfiguriert und eingerichtet. Dies beinhaltet die Erstellung und ggf. die Bearbeitung der Datenbankdateien sowie der Tabellen. Abschließend wird im Sicherheitsmanagement ein Benutzer angelegt, welcher auf die bestehende sowie die neue Datenbasis Zugriff erhält.

Im Zuge der Inbetriebnahme hat sich gezeigt, dass einige Parameter der Serveranwendung dynamisch anpassbar sein sollten. Damit wird die Anwendung sehr variabel in Bezug auf ihre Umgebung. Neben klassischen Parametern wie IP-Adresse, Port-Nummern und Datenbank-Verbindungsdaten, wurde auch das in Kapitel 6.1 behandelte Zertifikatshandling dynamisiert. Dies beinhaltet auch, dass das Im-/Exportieren sowie das (Ent-)Binden des Zertifikats während der Laufzeit der Anwendung geschieht. Ein manuelles Einrichten beim Umzug des Wrappers auf einen anderen Server entfällt somit.

Vor dem ersten Start der Wrapper-Applikation wird geprüft, ob die gewünschten Adressräume bereits belegt sind. Die Konfigurationsdatei wird z.B. in Port-Nr. für *HTTP* (Port: 8080) und *HTTPS* (Port: 8443) entsprechend angepasst. Da die Anwendung im Vorfeld mittels der Debug-Funktionalität (dt. Fehler beseitigen) der *IDE* bereits schrittweise in Betrieb genommen wurde, kann Sie nun direkt gestartet werden. Status und ggf. Fehlermeldungen erscheinen im Ausgabefenster der Applikation. Mittels der Schaltfläche „Start Communication Manager“ beginnt die Einrichtung des Kommunikationskanals. Dazu wird u. a. die gewählte IP-Adresse gegen vorhandene Netzwerkadapter geprüft. Hat der „Communication Manager“ die Initialisierungsroutine durchlaufen, so befindet er sich im Stand-by (dt. Bereitschaft) und wartet auf Clientanfragen.

7.2 Qualitätssteigerung durch Tests

Ein Kunde vertraut auf die angepriesenen Eigenschaften bzw. Funktionen eines Produktes als Gegenwert seiner finanziellen Aufwendungen. Die Beschaffenheit dieser garantierten Merkmale ist ein Ausdruck ihrer Qualität. Dabei kann von einer hohen Qualität gesprochen werden, wenn die Eigenschaften bzw. Funktionen, die an sie gestellten Anforderungen vollständig erfüllen. Damit die Qualität eines Softwareproduktes kein Zufall

ist, muss diese im Vorfeld evaluiert werden. Hierfür gibt es in der Softwareentwicklung zwei gängige Praktiken - Fehlersuche und Funktionstest. Die Fehlersuche wird auch als „debuggen“ bezeichnet, wobei die Vorsilbe das gleichzeitige Entfernen des Fehlers impliziert. Dazu wird die gewünschte Funktion mit einem Datensatz geladen und dessen (Teil-)Ergebnis anschließend evaluiert. Bei einem Test ist die Herangehensweise eher destruktiv. Es werden im Vorfeld Eingangs- sowie Ergebnisparameter spezifiziert und ein Algorithmus für den Test festgelegt. Anschließend kann der Test mit den verschiedenen Parametern durchgeführt werden, wobei es nur zwei mögliche Ergebnisse gibt - bestanden oder nicht bestanden. Es gibt unterschiedliche Arten von Tests. Je nach Implementierung-/Integrationsstufe gilt es verschiedene Ziele zu überprüfen. Die anschließende Aufzählung soll einen Überblick über die möglichen Test liefern, welche im Laufe der Entwicklung iterativ sowie mehrfach durchlaufen werden.

1. Modultest (Unittest)
2. Integrationstest
3. Systemtest (explorativer Test, Performancetest)
4. Abnahmetest (Akzeptanztest)

Im zeitlichen Verlauf der Realisierung steigt dabei die Komplexität der Softwaretests. Dies soll an dem Beispiel „Aufbau einer Datenbankverbindung“ gezeigt werden. Der betreffende Modul- bzw. Unittest prüft die dafür zuständige Methode, wie Abbildung 7.1 zeigt. Dieser einfache Test zeigt deutlich das „*AAA*“-Prinzip, welches gerade bei Unittest angewendet wird.[24] Dabei wird zu Beginn der Test vorbereitet. Anschließend wird das zu testende Element ausgeführt bzw. bearbeitet und abschließend erfolgt der Vergleich mit der Annahme bzw. dem erwarteten Ergebnis.

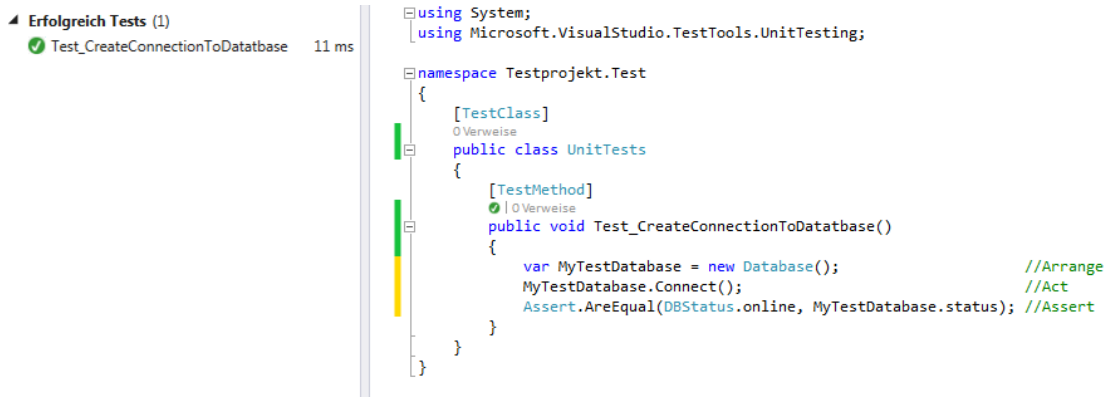

Abbildung 7.1: Bildschirmausschnitt der Testumgebung mit exemplarischem Unittest und dessen Ergebnis

Bei Integrationstests wird die Zusammenarbeit von mehreren Modulen geprüft. Wird hingegen die komplette Anwendung mit all ihren Modulen getestet, so spricht man von

[24] Vgl. Gärtner 2013, S.2

Systemtests. Um das Wrappersystem auch ohne Drittanwendung zu testen, wurde eine Testumgebung namens „Wrapper Test “ geschaffen. Mit diesem Werkzeug kann die Kommunikation zwischen dem Wrapper und einem vermeintlichen Client analysiert werden. Hierzu können *XML*-Requests abgesetzt und *XML*-Responses empfangen werden. Dies ermöglicht eine unmittelbare Fehlersuche sowie die Aufdeckung von Schwachstellen des Systems. Neben Einstellmöglichkeiten für den Verbindungsaufbau bietet die Software auch Zusatzinformationen zur Datenübertragung. Dies beinhaltet u. a. die Anzahl der gesendeten bzw. empfangenen Zeichen sowie die Reaktionszeit des Servers. Abbildung 7.2 zeigt die Benutzeroberfläche von „Wrapper Test“ mit einem einfachen Beispiel.

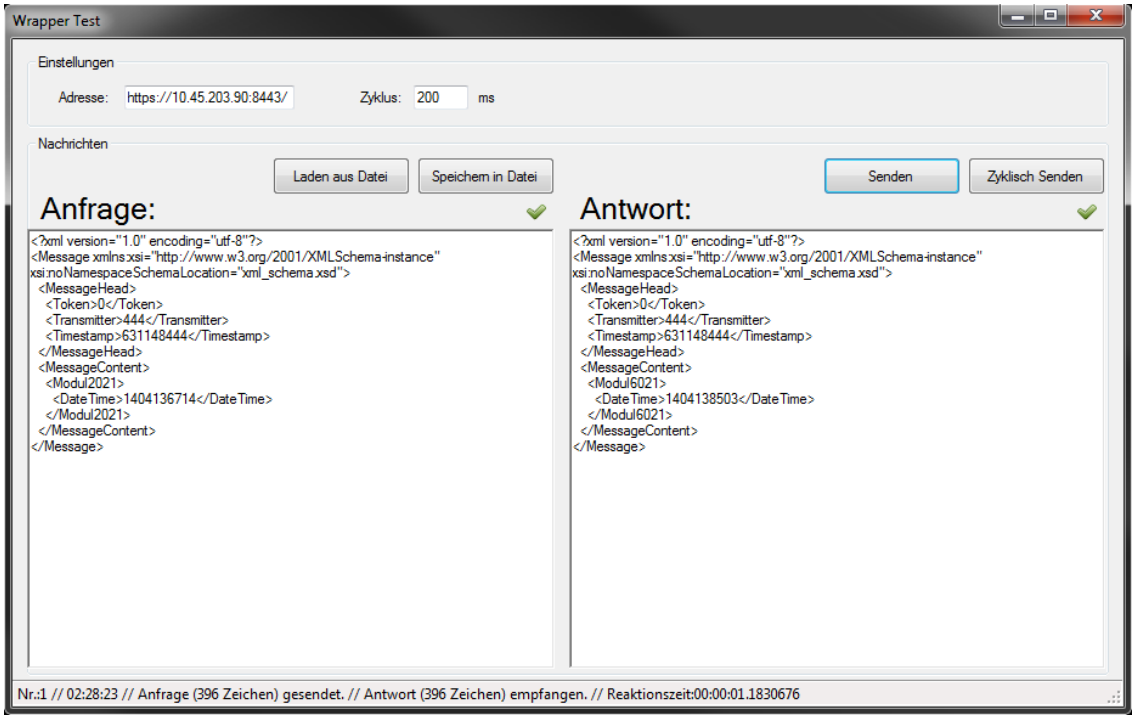

Abbildung 7.2: Benutzeroberfläche der Software „Wrapper Test“ zur Simulation von Clientanfragen

Um eine Nachricht mit diesem Werkzeug an die Vermittler-Software zu senden, hat der Tester drei Möglichkeiten. Entweder gibt er die vollständige Nachricht manuell ein oder lässt sich das Nachrichtengerüst automatisch generieren und ergänzt dieses. Die komfortabelste Variante ist das Laden der kompletten Nachricht aus einer Datei. Ist die Nachricht vollständig, kann sie mit der Schaltfläche „Nachricht senden“ an der Wrapper abgeschickt werden. Dort wird sie wie eine gewöhnliche Clientanfrage, gemäß der in Kapitel 4.2 beschriebenen Verarbeitungskette, behandelt. Die Rückantwort wird als *XML*-Text dargestellt. Eine Deserialisierung findet nicht statt. Der Antwortnachricht können die angeforderten Informationen oder ggf. vom Wrapper generierte Fehlermeldungen entnommen werden. Der dargestellte Anwendungsfall in Abb. 7.2 zeigt das Anfragemodul 2021 sowie das zugehörige Antwortmodul 6021. In der Statuszeile am unteren Bildschirmrand wird eine Reaktionszeit von ca. einer Sekunden bei einer übertragenen

Zeichenanzahl von jeweils 396 Zeichen angezeigt. Mittels diesem Systemtest konnte zwar die angesprochene Datenbankverbindung ebenfalls geprüft werden, allerdings müssen für dessen erfolgreiche Absolvierung auch noch andere Bedingungen erfüllt sein.

Ein Abnahmetest für den Wrapper alleine wäre nicht dienlich. Erst durch das Zusammenspiel mit einer Drittanwendung ist dies sinnvoll. Der Verfasser hat parallel zum Wrapper eine solche Drittanwendung entwickelt. Somit konnten Akzeptanztests mit beiden Anwendungen durch die Stakeholder erfolgreich durchgeführt werden.

7.3 Rollout/Release

„Rollout“ (dt. ausrollen) bezeichnet die Einführung eines neuen Produktes bzw. Systems. Im betrachten Anwendungsfall geht es um die Veröffentlichung der Software. Der Wrapper wurde mit der Drittanwendung gemeinsam ins Feld geführt. Dabei wurde iterativ vorgegangen. Dies bietet im Gegensatz zur Einführungsart „Big Bang“ die Vorteile, dass zum einen immer nur ein begrenzter Funktionsumfang veröffentlicht wird und zum anderen der Nutzerkreis überschaubar bleibt, welcher mit neuen Funktionen konfrontiert wird. Für den Alphatest wurde ein computeraffiner Nutzerkreis ausgewählt. So konnten letzte Schwachstellen durch gezielte und aussagekräftige Rückmeldungen beseitigt werden. Dadurch konnte bereits im Beta-Status ein akzeptables Ergebnis mit minimiertem Funktionsumfang erzielten werden. Der nutzbare Funktionsumfang sowie der Nutzerkreis wurde sukzessive erweitert, wie Abbildung 7.3 zeigt. Das Handling mit Versionen und Patches (dt. Nachbesserung) wird durch die eingesetzte Versionsverwaltung „Subversion“ erleichtert.

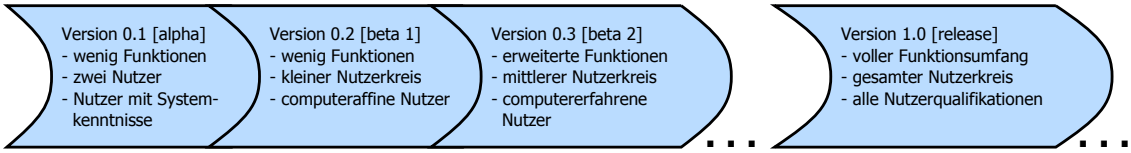

Abbildung 7.3: Verlauf der iterativen Softwareeinführung mittels steigendem Funktionsumfang sowie Nutzerkreis bei sinkender Nutzerqualifikation

Vor dem Rollout wurde der Nutzerkreis durch eine Schulung über den Funktionsumfang und dessen Nutzung unterrichtet. Da im Normalfall der Wrapper für den Nutzer nicht sicht-/spürbar ist, enthielt die Schulung nur Themengebiete der Drittanwendung, welche in Abbildung 8.1 dargestellt ist.

7.4 Fehlerbetrachtung und Angriffsszenarien

In der Industrie sind maximal 0,3 - 2 Fehler pro 1000 Zeilen Quellcode gefordert.[25] Mittels Alpha- und Beta-Test wurde die Fehleranzahl während der Entwicklungszeit und der Feldeinführung reduziert. Als Software, welche mit anderen Anwendungen kommuniziert, vervielfacht sich allerdings das Fehlerpotential. Die Fehlerbetrachtung in Tabelle 7.1 zeigt einen Auszug potenzieller Schwierigkeiten sowie Fehler, die dennoch auftreten können, und die folgende bzw. gezielte Reaktion der Anwendung darauf.

Quelle	Schwierigkeit/Fehler	Reaktion
Datenbank	Anw. kann Antwort von Datenbank nicht verarbeiten	Informationsverarbeitung wird abgebrochen
	Datenbank nicht erreichbar	Informationsverarbeitung wird abgebrochen
Client	gleichzeitige Anfragen	werden sequentiell abgearbeitet
	anfragender Client nicht mehr erreichbar	Antwortnachricht wird verworfen
	Anfrage mit gefährlichem Inhalt	Injektionsprüfung fängt Nachricht ab
	Anfrage mit falschem Inhalt	Berechtigungsprüfung fängt Nachricht ab
	Anfrage mit unzureichender Berechtigung	Berechtigungsprüfung fängt Nachricht ab
	Anfrage unvollständig oder Semantik falsch	Validierung oder Deserialisierung fängt Nachricht ab
Wrapper	Anw. stürtzt ab (keine Rückmeldung)	Anw. muss manuell neu gestartet werden
	Client erreicht Anw. nicht	Fehlerhandling (z. B. Timeout) muss clientseitig erfolgen
	sendet falschen SQL-Query an Datenbank	Informationsverarbeitung wird abgebrochen
	Anw. mehrfach gestartet	gleiche IP/Port ($\rightarrow$ nicht möglich) oder unterschiedliche IP/Port ($\rightarrow$ möglich)
	Anw. ist falsch konfiguriert	unterschiedlich ggf. kein Start möglich

Tabelle 7.1: Auszug aus der Fehlerbetrachtung der Software „Wrapper“ nach (Fehler-)Quellen unterteilt

Die Fehler bzw. deren Reaktionen enden i. A. in einem Abbruch der Anfragebearbeitung. Die Nichterfüllung der Arbeitsaufgabe wird dem Client mittels spezifischem „Modul Error“ in der Antwortnachricht übermittelt. Die Auswertung dieser aufbereiteten Informationen obliegt der Drittanwendung. Weitere Status- oder Fehlerinformationen können dem Ausgabefeld Wrapper-*GUI* entnommen werden.

[25] Vgl. Werner 2008, S.10

Neben der Betrachtung potenzieller Fehlerquellen sollen in diesem Kapitel auch Schwachstellen im Sinne der Angreifbarkeit aufgezeigt werden. Hierzu bildet Tabelle 7.2 eine Übersicht über gängige Angriffsszenarien, ihre Bedeutung, das Gefährdungsrisiko sowie die eingesetzten Abwehrmechanismen.

Szenario	Erläuterung	Risiko	Abwehrmechanismus
Sniffer	Informationsgewinn durch Mithören des Netzwerkverkehrs	sehr gering	Verschlüsselung mittels *HTTPS*
Man-in-the-middle / Phishing	Zwischenschalten eines Proxys zum Mithören und Verändern von Informationen	gering	Verwendung von Zertifikaten zur Authifizierung
Denial-of-service	Dienstverweigerung aufgrund Überlastung des Servers	gering	keine
Password-brute-force	Passwortidentifizierung mittels Tests aller möglichen Passwortkombinationen	gering	keine
Password-rainbow-table	Passwortidentifizierung in der Datenbank mittels Rückführung auf gebräuchliche Passwortkombinationen	sehr gering	Passwörter nicht klassisch MD5 gehasht

Tabelle 7.2: Angriffsszenarien und Abwehrmechanismen der Software „Wrapper“

Das geringe Gefährdungsrisiko der verschiedenen Angriffsszenarien liegt u. a. in der verwendeten Netzwerkstruktur begründet. Weiterhin wird davon ausgegangen, dass der Server auf dem die Wrapper-Anwendung läuft für einen Angreifer nicht zugänglich ist. Außerdem stellt nach Meinung des Verfassers die Software ein eher unattraktives Angriffsziel dar, da mittels der Wrapper-Anwendung kein Zugriff auf erweiterte Personal- oder Kundendaten möglich ist. Trotzdem sind die eingesetzten Abwehrmechanismen sinnvoll, um ggf. den Einstieg zu erschweren. Soll mit einer Drittanwendung über eine „unsichere“ Infrastruktur, wie das Internet, kommuniziert werden, so ist eine erneute Prüfung ratsam.

8 Zusammenfassung

8.1 Resümee

Enterprise-Resource-Planning-Systeme sind in Unternehmen heute nicht mehr wegzudenken. Sie vereinfachen Verwaltungsaufgaben, unterstützen die Buchhaltung und liefern adhoc aktuelle Kennzahlen. Dies liegt nicht nur in der Speicherung und Organistation unzähliger Datenmengen begründet. Auch die Vernetzung aller Mitarbeiter zur Erhebung dieser Daten spielt eine wichtige Rolle. Angesichts der Vereinfachung solcher Systeme, durch die Nutzung intuitiv bedienbarer Applikationen am Computerarbeitsplatz oder handlicher Terminals in der Montagehalle, ist ein weiterer Schritt in Richtung transparenter Workflows in der Projektarbeit getan.

Die Ergebnisse dieser Arbeit sind Teil einer solchen Applikation. Es wurde eine Vermittler-Software entwickelt, die einen bestehenden Datenhaushalt auswertet, aktualisiert und erweitert. Die entstandene Anwendung vermittelt zwischen einer beliebigen Drittanwendung und dem besagten Datenhaushalt. Dadurch wird es möglich auf das bereits existierende System mit einer zeitgemäßen Clientapplikation zuzugreifen. Durch die entstandene Kommunikationsschnittstelle, welche an aktuellen Standards ausgerichtet ist und weit verbreitete Protokolle einsetzt, können Drittanwendungen mit geringem Aufwand implementiert und integriert werden. Außerdem liefert es die Möglichkeit gängige unternehmenseigene Kennziffern und Bezeichner in dieser Plattform zu verwenden.

Neben den allgemeinen Anforderungen konnten auch die funktionalen umgesetzt werden. Dabei war eine besondere Herausforderung, ob das bestehende System neue Datenpakete von der Vermittlungssoftware akzeptiert und verarbeitet. Dies ist erfolgreich gelungen und ermöglicht eine Ablösung der z. Z. eingesetzten Clientapplikation. Der Funktionsumfang der vorangegangenen Anwendung konnte bereits schon komplett nachgebildet und ins Feld geführt werden. Weitere Funktionen werden folgen. Während der Designreviews (dt. Entwurfsprüfungen) wurde positives Feedback von den Stakeholdern zur enstandenen Lösungen signalisiert. Dies manifestierte sich bezogen auf das Endprodukt.

8.2 Vor- und Nachteile

Die entstandene Lösung wird in Zusammenhang mit einer Drittanwendung die bis dato genutzte Clientapplikation vollständig ersetzen. Die derzeitige Software ist unter dem aktuellen Betriebssystem nicht mehr lauffähig und musste mittels Virtualisierung allen Mitarbeitern zugänglich gemacht werden. Dies führte zu einer aufwendigen Prozedur um die Anwendung zu nutzen. Im neuen System kann sowohl der Wrapper als auch die

Drittanwendung auf einem Server in der vorhandenen IT-Struktur abgelegt werden. Der Nutzer kann die Clientapplikation direkt starten ohne dass eine Installation nötig wird. Dies schont lokale Ressourcen und vereinfacht die Wartung. Aufgrund einer begrenzt verfügbaren Anzahl von Lizenzen wurde man bei der alten Clientanwendung nach kurzer Zeitdauer automatisch abgemeldet. Das neue System arbeitet lizenzfrei, sodass es zu keinen unerwünschten Abmeldungen mehr kommt.

Die einzelnen Funktionalitäten sind in Module gekapselt. Durch die Aneinanderreihung dieser Module können diverse Funktionen miteinander kombiniert werden. Auf Grundlage dieses Universalbausatzes konnte bereits für die aktuelle Entwicklung der Drittanwendung eine Wrapper-*API* implementiert werden. Außerdem ist durch den modularen Aufbau eine Erweiterung mit geringem Aufwand jederzeit möglich. Dies macht das System für zukünftige Herausforderungen leistungsfähig.

Erster Ansatzpunkt ist dabei die Abbildung firmeneigener Prozesse. Ob es das Einreichen des Urlaubsantrages oder die nachträgliche Erfassung von Dienstreisen ist - all dies kann mit dem neuen System realisiert werden. Durch die hausinterne Realisierung dieses Projektes ist man zudem unabhängig gegenüber Dritten, z. B. bei der Nachrüstung von Funktionen. Dies hat die Folge, dass sich i. d. R. die Umsetzungsdauer und -kosten verringern. Ein weiterer Vorteil ist das eingesetzte Protokoll *HTTPS*. Es liefert eine verschlüsselte Verbindung zwischen Clientapplikation und dem Wrapper. Der Wrapper selbst ist auf dem zentralen Zeiterfassungsserver positioniert, welcher als „sicher“ eingestuft wird. Somit ist die gesamte Verarbeitungskette geschützt.

Aufgrund der indirekten Kommunikation infolge des geforderten Kommunikationsprinzips, siehe Abbildung 2.3, hat sich allerdings der Informationsweg zwischen Clientapplikation und Datenhaushalt verlängert. Der Weg wurde um zwei Transport- sowie zwei Verarbeitungseinheiten vergrößert. Da die Reaktionszeit abhängig vom Transportweg sowie den Verarbeitungsschritten ist, verlängert sich diese. Bei der Integration der neuen Clientanwendung zeigt sich, dass der Leistungsverlust vertretbar ist. Das Starten der Clientanwendung dauert ca. 2 Sekunden. Nach maximal weiteren 3 Sekunden ist das automatische Laden der Module für die erste Menüansicht der *GUI* abgeschlossen.

8.3 Ausblick

Durch den geschaffenen Vermittler ist ein Zugang zur existierenden Avero-Datenbank entstanden. Dieser Zugang kann zukünftig genutzt werden, um den Datenbestand zu pflegen und zu erweitern. Dazu wurde parallel eine Clientapplikation entwickelt, deren Oberfläche in Abbildung 8.1 dargestellt ist. Eine solche Anwendung bietet den Nutzern die gewohnten Funktionalitäten in einer auf ihre Bedürfnisse angepassten Arbeitsumgebung.

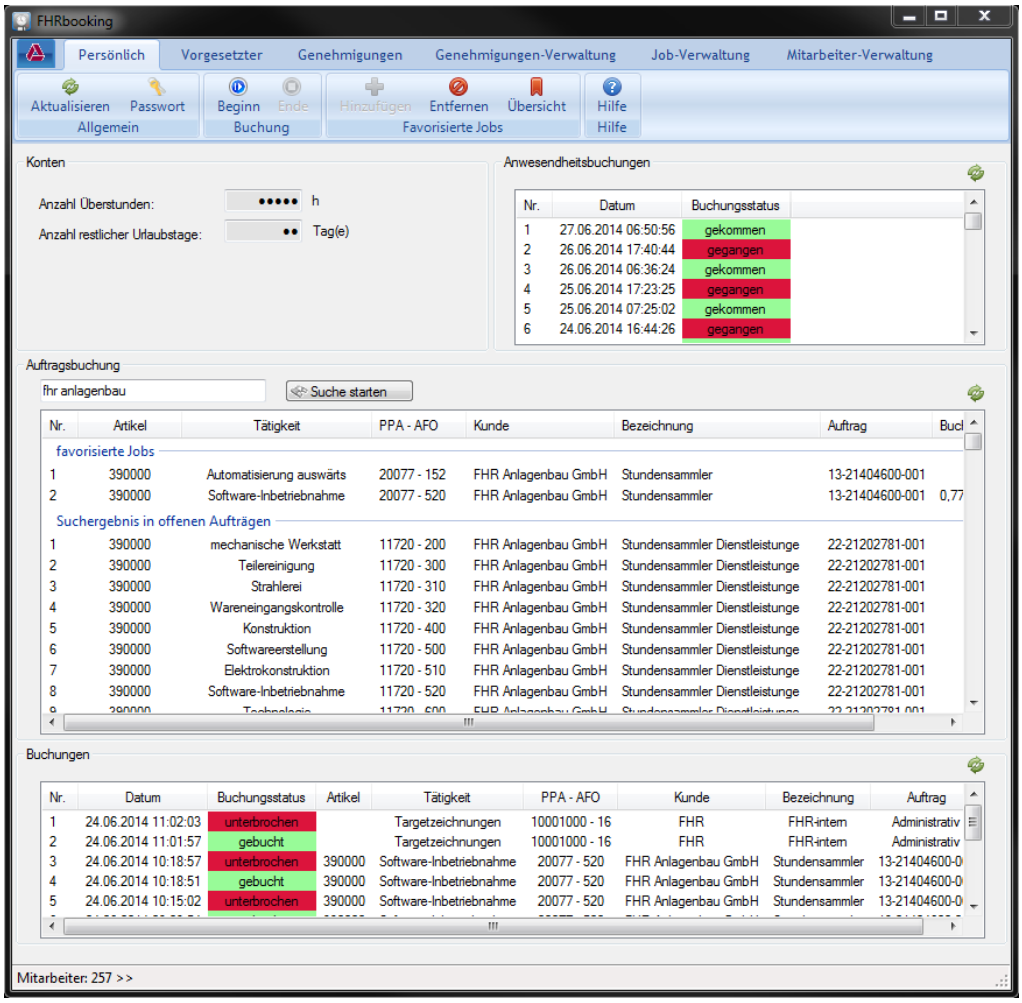

Abbildung 8.1: Benutzeroberfläche einer beispielhaften Clientapplikation

Neben einer Clientapplikation für das lokale Netzwerk ist auch die Realisierung einer Webapplikation für den Fernzugriff denkbar. Dies würde den bestehenden Informationshaushalt weltweit verfügbar machen. Somit wäre man auch auf einer Dienstreise mit dem System verbunden.

Weiterhin sind auch andere Schnittstellen zu weiteren Systemen denkbar. So könnte ein Mitarbeiter automatisch auf ein Projekt gebucht werden, wenn er dieses aktuell in einem der eingesetzten *CAD*-Programme bearbeitet. Ferner könnte der Funktionsumfang auch im Bereich der firmeneigenen Auswertungen wachsen. Diesbezüglich könnten Informationen generiert werden, welche es der Buchhaltungs- bzw. Controllingabteilung ermöglichen detailliertere Rückschlüsse auf die Projektarbeit zu schließen. Die gewonnenen Erkenntnisse daraus könnten in konkrete Projektvorgaben fließen.

Anhang A: Tabellen

A.1 Handlungszwänge der Akteure

Akteure	Nutzer-Handlung	Wrapper-Aktivitäten
Mitarbeiter	Gleitzeitkonto anzeigen	Gleitzeitkontostand abfragen
Mitarbeiter	Resturlaub anzeigen	Resturlaub abfragen
Mitarbeiter	Urlaub beantragen	Genehmigungsanfrage Urlaub stellen/bearbeiten
Mitarbeiter	Überstunden absetzen beantragen	Genehmigungsanfrage Stunden stellen/bearbeiten
Mitarbeiter	Dientzeiten anzeigen	Dienstbuchungen anzeigen
Mitarbeiter	erfolgte Buchungen anzeigen	Buchungen anzeigen
Mitarbeiter	aktuelle Buchungen anzeigen	aktuell laufende Jobs anzeigen
Mitarbeiter	offene Jobs anzeigen	Jobs anzeigen
Mitarbeiter	gebuchte Stunden auf Job anzeigen	gebuchte Stunden anzeigen
Mitarbeiter	Job-Favoriten-Liste anzeigen	Job-Favoriten-Liste anzeigen
Mitarbeiter	Einträge zu Job-Favoriten-Liste hinzufügen	Jobs anzeigen, Job-Favoriten-Liste anzeigen, Job-Favoriten-Liste bearbeiten
Mitarbeiter	Einträge von der Job-Favoriten-Liste entfernen	Job-Favoriten-Liste anzeigen, Job-Favoriten-Liste bearbeiten
Mitarbeiter	nachträglich Arbeitsbeginn/-ende einreichen	Genehmigungsanfragen anzeigen, Genehmigungsanfrage Dienstbeginn stellen/bearbeiten, Genehmigungsanfrage Dienstende stellen/bearbeiten
Mitarbeiter	nachträgliche Buchung absetzen	Jobs anzeigen, aktuell laufende Jobs anzeigen, Genehmigungsanfragen anzeigen, Job-Favoriten-Liste anzeigen, Genehmigungsanfrage Buchung stellen/bearbeiten
Mitarbeiter	Buchung auf offenen Job absetzen	Jobs anzeigen, Buchungen auf offenen Job durchführen, aktuell laufende Jobs anzeigen, Job-Favoriten-Liste anzeigen
Mitarbeiter	an der Anwendung anmelden	An-/Abmelden

Mitarbeiter	an der Anwendung abmelden	An-/Abmelden
Mitarbeiter	Passwort ändern	Mitarbeiterpasswort setzen
Mitarbeiter	eigene Genehmigungsanfragen einsehen	Genehmigungsanfragen anzeigen
Mitarbeiter	eigene Genehmigungsanfrage bearbeiten	Genehmigungsanfragen anzeigen, Genehmigungsanfrage Urlaub stellen/bearbeiten, Genehmigungsanfrage Stunden stellen/bearbeiten, Genehmigungsanfrage Dienstbeginn stellen/bearbeiten, Genehmigungsanfrage Dienstende stellen/bearbeiten, Genehmigungsanfrage Buchung stellen/bearbeiten
Mitarbeiter	eigene Genehmigungen entfernen	Genehmigungsanfragen anzeigen, Genehmigungsanfrage entfernen
Vorgesetzer	Resturlaub untergeordneter Mitarbeiter anzeigen	Resturlaub abfragen, Mitarbeiter anzeigen
Vorgesetzer	Gleitzeitkonto untergeordneter Mitarbeiter anzeigen	Gleitzeitkontostand abfragen, Mitarbeiter anzeigen
Vorgesetzer	gebuchte Stunden untergeordneter Mitarbeiter anzeigen	gebuchte Stunden anzeigen, Mitarbeiter anzeigen
Genehmiger	Genehmigungsanfragen der untergeordneten Mitarbeiter einsehen	Genehmigungsanfragen anzeigen, Mitarbeiter anzeigen
Genehmiger	Genehmigungsanfrage genehmigen	Genehmigungsanfragen anzeigen, Genehmigungsstatus bearbeiten
Genehmiger	Genehmigungsanfrage ablehnen	Genehmigungsanfragen anzeigen, Genehmigungsstatus bearbeiten
Genehmiger	Genehmigungsrecht- weiterleitungen anzeigen	Genehmigungsweiterleitungen anzeigen
Genehmiger	Genehmigungsrecht weiterleiten	Genehmigungsweiterleitungen bearbeiten, Genehmigungsweiterleitungen anzeigen, Mitarbeiter anzeigen
Genehmiger	Genehmigungsrecht- weiterleitung entziehen	Genehmigungsweiterleitungen bearbeiten, Genehmigungsweiterleitungen anzeigen
Genehmigungsverwalter	genehmigte Genehmigungsanfragen einsehen	Genehmigungsanfragen anzeigen

Genehmigungsverwalter	genehmigte Genehmigungsanfragen als erledigt markieren	Genehmigungsanfragen anzeigen, Genehmigungsstatus bearbeiten
Genehmigungsverwalter	genehmigte Genehmigungsanfragen als abgelehnt markieren	Genehmigungsanfragen anzeigen, Genehmigungsstatus bearbeiten
Jobverwalter	Jobs anzeigen	Jobs anzeigen
Jobverwalter	alle Jobs importieren	Job-Daten importieren
Jobverwalter	alle Jobs einer Rückmeldenummer importieren	Job-Daten importieren
Jobverwalter	gebuchte Stunden auf Job anzeigen	gebuchte Stunden anzeigen
Nutzerverwalter	administrative Mitarbeiterinformationen einsehen	Mitarbeiter anzeigen, Mitarbeitereinstellungen anzeigen
Nutzerverwalter	neuen Mitarbeiter zur Anwendung hinzufügen	Mitarbeiter anzeigen, Mitarbeitereinstellungen bearbeiten
Nutzerverwalter	bestehenden Mitarbeiter der Anwendung bearbeiten	Mitarbeiter anzeigen, Mitarbeitereinstellungen anzeigen, Mitarbeitereinstellungen bearbeiten
Nutzerverwalter	Mitarbeiter aus der Anwendung entfernen	Genehmigungsanfragen anzeigen, Genehmigungsrecht weiterleiten/entziehen, Genehmigungsweiterleitungen anzeigen, Mitarbeiter anzeigen, Mitarbeitereinstellungen bearbeiten, Vorgesetztenrecht abgeben
Nutzerverwalter	Mitarbeiterpasswort ändern	Mitarbeiterpasswort setzen

Tabelle A.1: Übersicht der Akteure ihrer Handlungszwänge sowie die diesbezüglichen Wrapper-Einzelaktivitäten

A.2 Auflistung aller Berechtigungen

ID	Bezeichner	Beschreibung
V01	validMe	prüft ob man Absender ist
V02	validSuperior	prüft ob man Vorgesetzer ist
V03	validEmployee	prüft ob es Mitarbeiter gibt
V04	validJobAdmin	prüft ob man Job-Verwalter ist
V05	validJob	prüft ob Job vorhanden ist
V06	validLoc	prüft wie man angemeldet ist
V07	validToken	prüft ob Token gültig ist
V08	validJobOpen	prüft ob Auftrag offen ist
V09	validApprove	prüft ob man für Person genehmigen darf (entweder Genehmiger oder dessen Vertretung)
V10	validApprovalNumber	prüft ob die Genehmigungsnummer zur Person passt
V11	validPassword	prüft ob Passwort gültig ist
V12	validProxyApprover	prüft ob ein bestimmter Genehmigungsvertreter vorhanden ist
V13	validTimestampFuture	prüft ob ein Zeitpunkt in der Zukunft liegt
V14	validJobBooked	prüft ob man auf bestimmten Auftrag gebucht ist
V15	validFavoritJob	prüft ob man einen bestimmten FavoritJob hat
V16	validUserMe	prüft ob der Benutzername gültig ist
V17	validUserAdmin	prüft ob man Nutzer-Verwalter ist
V18	validApprovalAdmin	prüft ob man Genehmigung-Verwalter ist
V19	validApprovalDelete	prüft ob eine Genehmigungsanfrage gelöscht werden kann

Tabelle A.2: Auflistung aller Berechtigungen

A.3 Übersicht Wrapper-Module

Aktivität	Request-Nr	Response-Nr	Prüfungen
Gleitzeitkontostand abfragen	1001	5001	validToken, validEmployee, validMe, validSuperior
Resturlaub abfragen	1002	5002	validToken, validEmployee, validMe, validSuperior
Buchungen anzeigen	1003	5003	validToken, validEmployee, validMe, validSuperior
Jobs anzeigen	1004	5004	validToken, validEmployee, validJobAdmin
gebuchte Stunden anzeigen	1005	5005	validToken, validEmployee, validMe, validSuperior, validJobAdmin, validJob
Buchungen auf offenen Job durchführen	1006	5006	validToken, validEmployee, validJob, validJobOpen, validJobBooked
Dienstbuchungen anzeigen	1007	5007	validToken, validEmployee, validMe
aktuell laufende Jobs anzeigen	1008	5008	validToken, validEmployee, validEmployee, validMe, validSuperior
*** ohne Funktion ***	2001	6001	
Genehmigungsanfragen anzeigen	2002	6002	validToken, validEmployee,validMe, validApprove, validApprovalAdmin
Genehmigungsweiterleitungen bearbeiten	2003	6003	validToken, validMe, validEmployee, validProxyApprove
Genehmigungsweiterleitungen anzeigen	2004	6004	validToken, validEmployee, validEmployee, validMe
geplante Buchung abspeichern	2005	6005	validToken, validEmployee, validJob, validJobOpen, validTimestampFuture, validJobBooked
Job-Favoriten-Liste anzeigen	2006	6006	validToken, validEmployee, validMe, validUserAdmin
Job-Favoriten-Liste bearbeiten	2007	6007	validToken, validEmployee, validEmployee, validMe, validUserAdmin, validFavJob

An-/Abmelden	2008	6008	validToken, validEmployee, validMe, validPassword
Mitarbeiter anzeigen	2009	6009	validToken, validEmployee, validMe, validUserAdmin
Anfrage Zusatzinformationen anzeigen	2010	6010	ValidToken, validEmployee, validMe, validApprove, validApprovalAdmin
Anfrage Zusatzinformationen erstellen/bearbeiten	2011	6011	validToken, validEmployee, validApprovalNumber, vaiidJob
Genehmigungsstatus bearbeiten	2012	6012	validToken, validEmployee, validApprovalNumber, validApprover, validApprovalAdmin
Genehmigungsanfrage löschen	2013	6013	validToken, validEmployee, validApprovalNumber
Genehmigungsanfrage Urlaub stellen/bearbeiten	2014	6014	validToken, validEmployee, validApprovalNumber
Genehmigungsanfrage Stunden stellen/bearbeiten	2015	6015	validToken, validEmployee, validApprovalNumber
Genehmigungsanfrage Dienstbuchung	2016	6016	validToken, validEmployee, validApprovalNumber
*** ohne Funktion ***	2017	6017	
Genehmigungsanfrage Buchung stellen/bearbeiten	2018	6018	validToken, validEmployee, validApprovalNumber
Mitarbeitereinstellungen anzeigen	2019	6019	validToken, validEmployee, validUserAdmin,
Mitarbeitereinstellungen bearbeiten	2020	6020	validToken, validEmployee, validUserAdmin
Verbindungstest	2021	6021	
Vorgesetztenrecht abgeben	2022	6022	validToken, validEmployee, validUserAdmin,
Job-Daten importieren	2023	6023	validToken, validEmployee,validMe, validJobAdmin

Tabelle A.3: Übersicht der Wrapper-Module

Anhang B: Sonstiges

B.1 Glossar zur Geschäftslogik

Mitarbeiter-Name: Name des Mitarbeiters bestehend aus Familien- und Vorname; z. B. „Mustermann, Max“

Personalnummer: eindeutige Identifizierung eines Mitarbeiters; z. B. „842“

Mitarbeiter: besitzt Personalnummer und Mitarbeitername;

Überstunden: aktuelle Anzahl von Überstunden des Mitarbeiter; z. B. „55,3“

Urlaubstage: aktuelle Anzahl der Urlaubstage des Mitarbeiters; z. B. „20“

Mitarbeiter-Konto: besteht aus Personal-Nummer, Urlaubstagen und Überstunden eines Mitarbeiters; z. B. „842“,“20“,“55,3“

Vorgesetzter: Mitarbeiter, welcher einen oder mehrere Mitarbeiter führt

Zeitstempel: Angabe besteht i. d. R. aus Datum und Uhrzeit; bei der Übermittlung wird die Unixzeitdefinition verwendet; z.B. „2014-02-27 12:33:11“

Dienstbeginn: Buchungsart bei einer Dienstbuchung, wenn der Mitarbeiter i. A. seine Arbeit aufnimmt

Dienstende: Buchungsart bei einer Dienstbuchung, wenn der Mitarbeiter i. A. seine Arbeit unterbricht

Dienstbuchungsart: kann entweder „Dienstbeginn“ oder „Dienstende“ sein

Dienstbuchung: Verbuchung des Dienstbeginns oder -ende eines Mitarbeiters, somit keine (Job-)Buchung im eigentlichen Sinne; enthält Personalnummer, Dienstbuchungsart, Zeitstempel

AFO-Name: Benamung einer Arbeitsfolge bzw. Tätigkeit in einem Projekt; z.B. „Software-Erstellung“

AFO-Nummer: eindeutige Identifikation einer Arbeitsfolge bzw. Tätigkeit in einem Projekt; z. B. „0520“

AFO: Arbeitsfolge bzw. Tätigkeit in einem Projekt; besteht aus AFO-Nummer und AFO-Name; z. B. „0520“,„Software-Erstellung“

Rückmeldenummer: eindeutige Identifizierung eines Projektes; z.B. „000012345“

Zeichnungsnummer: Zusatzinformation zur Zeichnungsnummer eines Projekt; z. B. „1B0660-000-00000“

Auftragsnummer: Zusatzinformation zur Auftragsnummer eines Projektes; z. B. „11-123456789-123“

Kunde: Zusatzinformation zum Kunden eines Projektes; z. B. „Max Mustermann AG“

Bezeichnung: Zusatzinformation zur Bezeichnung eines Projektes; z. B. „FHR.Line.700“

Projekt: kann Zusatzinformationen wie Rückmeldenummer, Zeichnungsnummer, Auftragsnummer, Kunde, Bezeichnung enthalten

Job: Kombination aus Rückmeldenummer und AFO-Nummer; ist offen oder geschlossen;kann Projekt-Zusatzinformation enthalten; z. B. „0000123450520“

geschlossener Job: Job, dessen Rückmeldenummer in der Proalpha-Datenbank als „archiviert“ gekennzeichnet ist; auf diese Jobart darf nicht gebucht werden

offener Job: ist ein Job, dessen Rückmeldenummer in der Proalpha-Datenbank als „freigegeben“ gekennzeichnet ist; auf diese Jobart darf gebucht werden

Arbeitsbeginn: Buchungsart bei einer (Job-)Buchung, wenn der Mitarbeiter i. A. sein Arbeit an einem konkreten Job aufnimmt

Arbeitsende: Buchungsart bei einer (Job-)Buchung, wenn der Mitarbeiter i. A. seine Arbeit an einem konkreten Job unterbricht

Buchungsart: kann entweder „Arbeitsbeginn“ oder „Arbeitsende“ sein

(Job-)Buchung: Verbuchung des Arbeitsbeginns oder -ende an einem konkreten Job; enthält Job, Personalnummer, Buchungsart (, Zeitstempel)

Genehmigungsanfrage: Anfrage kann vom Typ „Urlaub“, „Überstunden absetzen“, „nachträgliche Zeiterfassung“ oder “nachträgliche Buchung“ sein

Genehmigungsanfrage Urlaub: Anfrage eines Mitarbeiters zur Bewilligung von Urlaub

Genehmigungsanfrage Überstunden absetzen: Anfrage eines Mitarbeiter zum Absetzen von Überstunden

Genehmigungsanfrage nachträgliche Zeiterfassung: Anfrage eines Mitarbeiter zur nachträglichen Zeiterfassung

Genehmigungsanfrage nachträgliche Buchung: Anfrage eines Mitarbeiter für nachträgliche Buchung

Genehmiger: Mitarbeiter, welcher Genehmigungsanfragen von die ihm zugewiesen Mitarbeiter sieht und darüber entscheidet; kann keinen, einen oder mehrere Genehmigungsvertreter besitzen

Genehmigungsvertreter: Mitarbeiter, welcher einen Genehmiger temporär vertritt und dessen Rechte erhält

Genehmigungen-Verwalter: Mitarbeiter, welcher erteilte Genehmigungen prüft und verbucht

Mitarbeiter-Verwalter: Mitarbeiter, welcher Nutzer (= Mitarbeiter) der Applikation hinzufügt oder deren Eigenschaften bearbeitet

Job-Verwalter: Mitarbeiter, welcher erweiterte Einsicht in die Verwaltung und Status von Jobs besitzt

B.2 XML-Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Schemata -->

<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 attributeFormDefault="unqualified" elementFormDefault="qualified"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:sch="http://www.ascc.net/xml/schematron">
    <xs:element name="Message">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="MessageHead">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Token" type="TokenTyp"/>
                <xs:element name="Transmitter" type="PersNoTyp"/>
                <xs:element name="Timestamp" type="TimestampTyp"/>
              </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="MessageContent" type="MessageContentTyp"/>
      </xs:sequence>
      </xs:complexType>
    </xs:element>

<!-- Typdefinitionen Modul(-auswahl) -->

<xs:complexType name="MessageContentTyp">
  <xs:choice>
    <xs:element name="ModulError" type="ModulErrorTyp"/>
    <xs:element name="Modul1001" type="Modul1001Typ"/>
    <xs:element name="Modul1002" type="Modul1002Typ"/>
    <xs:element name="Modul1003" type="Modul1003Typ"/>
    <xs:element name="Modul1004" type="Modul1004Typ"/>
    <xs:element name="Modul1005" type="Modul1005Typ"/>
    <xs:element name="Modul1006" type="Modul1006Typ"/>
    <xs:element name="Modul1007" type="Modul1007Typ"/>
    <xs:element name="Modul1008" type="Modul1008Typ"/>
    <xs:element name="Modul1009" type="Modul1009Typ"/>
    <xs:element name="Modul1010" type="Modul1010Typ"/>
```

```
<xs:element name="Modul2002" type="Modul2002Typ"/>
<xs:element name="Modul2003" type="Modul2003Typ"/>
<xs:element name="Modul2004" type="Modul2004Typ"/>
<xs:element name="Modul2005" type="Modul2005Typ"/>
<xs:element name="Modul2006" type="Modul2006Typ"/>
<xs:element name="Modul2007" type="Modul2007Typ"/>
<xs:element name="Modul2008" type="Modul2008Typ"/>
<xs:element name="Modul2009" type="Modul2009Typ"/>
<xs:element name="Modul2010" type="Modul2010Typ"/>
<xs:element name="Modul2011" type="Modul2011Typ"/>
<xs:element name="Modul2012" type="Modul2012Typ"/>
<xs:element name="Modul2013" type="Modul2013Typ"/>
<xs:element name="Modul2014" type="Modul2014Typ"/>
<xs:element name="Modul2015" type="Modul2015Typ"/>
<xs:element name="Modul2016" type="Modul2016Typ"/>
<xs:element name="Modul2018" type="Modul2018Typ"/>
<xs:element name="Modul2019" type="Modul2019Typ"/>
<xs:element name="Modul2020" type="Modul2020Typ"/>
<xs:element name="Modul2021" type="Modul2021Typ"/>
<xs:element name="Modul2022" type="Modul2022Typ"/>
<xs:element name="Modul2023" type="Modul2023Typ"/>
<xs:element name="Modul2024" type="Modul2024Typ"/>
<xs:element name="Modul5001" type="Modul5001Typ"/>
<xs:element name="Modul5002" type="Modul5002Typ"/>
<xs:element name="Modul5003" type="Modul5003Typ"/>
<xs:element name="Modul5004" type="Modul5004Typ"/>
<xs:element name="Modul5005" type="Modul5005Typ"/>
<xs:element name="Modul5006" type="Modul5006Typ"/>
<xs:element name="Modul5007" type="Modul5007Typ"/>
<xs:element name="Modul5008" type="Modul5008Typ"/>
<xs:element name="Modul5009" type="Modul5009Typ"/>
<xs:element name="Modul5010" type="Modul5010Typ"/>
<xs:element name="Modul6002" type="Modul6002Typ"/>
<xs:element name="Modul6003" type="Modul6003Typ"/>
<xs:element name="Modul6004" type="Modul6004Typ"/>
<xs:element name="Modul6006" type="Modul6006Typ"/>
<xs:element name="Modul6007" type="Modul6007Typ"/>
<xs:element name="Modul6008" type="Modul6008Typ"/>
<xs:element name="Modul6009" type="Modul6009Typ"/>
<xs:element name="Modul6010" type="Modul6010Typ"/>
<xs:element name="Modul6011" type="Modul6011Typ"/>
<xs:element name="Modul6012" type="Modul6012Typ"/>
<xs:element name="Modul6013" type="Modul6013Typ"/>
```

```
    <xs:element name="Modul6014" type="Modul6014Typ"/>
    <xs:element name="Modul6015" type="Modul6015Typ"/>
    <xs:element name="Modul6016" type="Modul6016Typ"/>
    <xs:element name="Modul6018" type="Modul6018Typ"/>
    <xs:element name="Modul6019" type="Modul6019Typ"/>
    <xs:element name="Modul6020" type="Modul6020Typ"/>
    <xs:element name="Modul6021" type="Modul6021Typ"/>
    <xs:element name="Modul6022" type="Modul6022Typ"/>
    <xs:element name="Modul6023" type="Modul6023Typ"/>
    <xs:element name="Modul6024" type="Modul6024Typ"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="ModulErrorTyp">
  <xs:sequence>
    <xs:element name="Error" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="ErrorLocation" type="TextTyp"/>
          <xs:element name="ErrorMessage" type="TextTyp"/>
          </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul1001Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul1002Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul1003Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="State" type="StateTyp"/>
    <xs:element name="DateStart" type="TimestampTyp"/>
```

```
    <xs:element name="DateEnd" type="TimestampTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul1004Typ">
 <xs:sequence>
    <xs:element name="State" type="StateTyp"/>
    <xs:element name="Search" type="SearchTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul1005Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="Job" type="JobTyp"/>
    <xs:element name="DateStart" type="TimestampTyp"/>
    <xs:element name="DateEnd" type="TimestampTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul1006Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="StateBooking" type="StateBookingTyp"/>
    <xs:element name="Job" type="JobTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul1007Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul1008Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul1009Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
```

```
    <xs:element name="DateStart" type="TimestampTyp"/>
    <xs:element name="DateEnd" type="TimestampTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul1010Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

  <!-- Modul 2001 ist leer -->

<xs:complexType name="Modul2002Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="State" type="StateTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2003Typ">
  <xs:sequence>
    <xs:element name="PersNoOwner" type="PersNoTyp"/>
    <xs:element name="PersNoProxy" type="PersNoTyp"/>
    <xs:element name="ApprovalForward" type="TaskTyp"/>
    <xs:element name="Fixed" type="YesNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2004Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2005Typ">
  <xs:sequence>
    <xs:element name="Date" type="TimestampTyp"/>
    <xs:element name="StateBooking" type="StateBookingTyp"/>
    <xs:element name="Job" type="JobTyp"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="Modul2006Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2007Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="JobFavoritTask" type="TaskTyp"/>
    <xs:element name="Job" type="JobTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2008Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="LogTask" type="LogTaskTyp"/>
    <xs:element name="Password" type="TextTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2009Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="Group" type="GroupTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2010Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2011Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="ApprovalBookingDataSet"
                type="ApprovalBookingDataSetTyp"/>
  </xs:sequence>
```

```
</xs:complexType>

<xs:complexType name="Modul2012Typ">
  <xs:sequence>
    <xs:element name="PersNoEmployee" type="PersNoTyp"/>
    <xs:element name="PersNoChanger" type="PersNoTyp"/>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
    <xs:element name="ApprovalStatus" type="ApprovalStatusTyp"/>
    <xs:element name="Comment" type="TextTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2013Typ">
  <xs:sequence>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2014Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
    <xs:element name="DateStart" type="TimestampTyp"/>
    <xs:element name="DateEnd" type="TimestampTyp"/>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="SpecialLeaveQuantity" type="QuantityTyp"/>
    <xs:element name="Reason" type="TextTyp"/>
    <xs:element name="Proxy" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2015Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
    <xs:element name="DateStart" type="TimestampTyp"/>
    <xs:element name="DateEnd" type="TimestampTyp"/>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="Proxy" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2016Typ">
```

```
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
    <xs:element name="DateStart" type="TimestampTyp"/>
    <xs:element name="DateEnd" type="TimestampTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2018Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
    <xs:element name="DateStart" type="TimestampTyp"/>
    <xs:element name="DateEnd" type="TimestampTyp"/>
    <xs:element name="Location" type="TextTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2019Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2020Typ">
  <xs:sequence>
    <xs:element name="Task" type="TaskTyp"/>
    <xs:element name="PersNoEmployee" type="PersNoTyp"/>
    <xs:element name="PersNoSuperior" type="PersNoTyp"/>
    <xs:element name="PersNoApprover" type="PersNoTyp"/>
    <xs:element name="StatusApprovalAdmin" type="YesNoTyp"/>
    <xs:element name="StatusJobAdmin" type="YesNoTyp"/>
    <xs:element name="StatusUserAdmin" type="YesNoTyp"/>
    <xs:element name="Password" type="TextTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2021Typ">
  <xs:sequence>
    <xs:element name="DateTime" type="TimestampTyp"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="Modul2022Typ">
  <xs:sequence>
    <xs:element name="PersNoOldSuperior" type="PersNoTyp"/>
    <xs:element name="PersNoNewSuperior" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2023Typ">
  <xs:sequence>
    <xs:element name="ShortJob" type="JobTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul2024Typ">
  <xs:sequence>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
    <xs:element name="Sort" type="ApprovalSortTyp"/>
  </xs:sequence>
</xs:complexType>


<xs:complexType name="Modul5001Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="Overtime" type="QuantityDecimalTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul5002Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="Holiday" type="QuantityTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul5003Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="State" type="StateTyp"/>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="BookingSet" type="BookingSetTyp"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="Modul5004Typ">
  <xs:sequence>
    <xs:element name="State" type="StateTyp"/>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="JobSet" type="JobSetTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul5005Typ">
  <xs:sequence>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="BookingHourSet" type="BookingHourSetTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul5006Typ">
  <xs:sequence>
    <xs:element name="DateTime" type="TimestampTyp"/>
    <xs:element name="StateBooking" type="StateBookingTyp"/>
    <xs:element name="Job" type="JobTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul5007Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="ArriveLeaveSet" type="ArriveLeaveSetTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul5008Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="CurrentJobSet" type="CurrentJobSetTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul5009Typ">
  <xs:sequence>
    <xs:element name="Quantity" type="QuantityTyp"/>
```

```
    <xs:element name="AbsenceSet" type="AbsenceSetTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul5010Typ">
  <xs:sequence>
    <xs:element name="Quantity" type="QuantityTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6002Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="State" type="StateTyp"/>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="ApprovalSet" type="ApprovalSetTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6003Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6004Typ">
  <xs:sequence>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="ProxyApproverSet"
                type="ProxyApproverSetTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6006Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="FavoritJobSet" type="FavoritJobSetTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6007Typ">
  <xs:sequence>
```

```
    <xs:element name="JobFavoritTask" type="TaskTyp"/>
    <xs:element name="Job" type="JobTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6008Typ">
  <xs:sequence>
    <xs:element name="LogTask" type="LogTaskTyp"/>
    <xs:element name="Token" type="TokenTyp"/>
    <xs:element name="StatusEmployee" type="YesNoTyp"/>
    <xs:element name="StatusSuperior" type="YesNoTyp"/>
    <xs:element name="StatusApproval" type="YesNoTyp"/>
    <xs:element name="StatusApprovalAdmin" type="YesNoTyp"/>
    <xs:element name="StatusJobAdmin" type="YesNoTyp"/>
    <xs:element name="StatusUserAdmin" type="YesNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6009Typ">
  <xs:sequence>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="EmployeeSet" type="EmployeeSetTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6010Typ">
  <xs:sequence>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="ApprovalBookingDataSet"
                type="ApprovalBookingDataSetTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6011Typ">
  <xs:sequence>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6012Typ">
  <xs:sequence>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
  </xs:sequence>
```

```
</xs:complexType>

<xs:complexType name="Modul6013Typ">
  <xs:sequence>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6014Typ">
  <xs:sequence>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6015Typ">
  <xs:sequence>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6016Typ">
  <xs:sequence>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6018Typ">
  <xs:sequence>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6019Typ">
  <xs:sequence>
    <xs:element name="Quantity" type="QuantityTyp"/>
    <xs:element name="UserSet" type="UserSetTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6020Typ">
  <xs:sequence>
    <xs:element name="PersNo" type="PersNoTyp"/>
    <xs:element name="Task" type="TaskTyp"/>
```

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

<xs:complexType name="Modul6021Typ">
  <xs:sequence>
    <xs:element name="DateTime" type="TimestampTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6022Typ">
  <xs:sequence>
    <xs:element name="PersNoNewSuperior" type="PersNoTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6023Typ">
  <xs:sequence>
    <xs:element name="QuantityImport" type="QuantityTyp"/>
    <xs:element name="QuantityFound" type="QuantityTyp"/>
    <xs:element name="QuantityNotFound" type="QuantityTyp"/>
    <xs:element name="QuantityAdd" type="QuantityTyp"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Modul6024Typ">
  <xs:sequence>
    <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
  </xs:sequence>
</xs:complexType>



  <!-- Arraydefinitionen -->

<xs:complexType name="BookingSetTyp">
  <xs:sequence>
    <xs:element name="Booking" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="DateTime" type="TimestampTyp"/>
          <xs:element name="StateBooking" type="StateBookingTyp"/>
          <xs:element name="Project" type="xs:string"/>
          <xs:element name="Description" type="xs:string"/>
```

```
          <xs:element name="Job" type="JobTyp"/>
          <xs:element name="Customer" type="xs:string"/>
          <xs:element name="Article" type="xs:string"/>
          <xs:element name="OperationDescription" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="JobSetTyp">
  <xs:sequence>
    <xs:element name="Job" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="StateTask" type="StateTaskTyp"/>
          <xs:element name="Project" type="xs:string"/>
          <xs:element name="Description" type="xs:string"/>
          <xs:element name="Job" type="JobTyp"/>
          <xs:element name="Customer" type="xs:string"/>
          <xs:element name="Article" type="xs:string"/>
          <xs:element name="OperationDescription" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="EmployeeSetTyp">
  <xs:sequence>
    <xs:element name="Employee" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="PersNo" type="PersNoTyp"/>
          <xs:element name="EmployeeName" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="UserSetTyp">
  <xs:sequence>
```

```
    <xs:element name="User" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="PersNoEmployee" type="PersNoTyp"/>
          <xs:element name="NameEmployee" type="xs:string"/>
          <xs:element name="PersNoSuperior" type="PersNoTyp"/>
          <xs:element name="NameSuperior" type="xs:string"/>
          <xs:element name="PersNoApprover" type="PersNoTyp"/>
          <xs:element name="NameApprover" type="xs:string"/>
          <xs:element name="StatusApprovalAdmin" type="YesNoTyp"/>
          <xs:element name="StatusJobAdmin" type="YesNoTyp"/>
          <xs:element name="StatusUserAdmin" type="YesNoTyp"/>
          <xs:element name="StatusLocation" type="LogTaskTyp"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ArriveLeaveSetTyp">
  <xs:sequence>
    <xs:element name="ArriveLeave" minOccurs="0"
                                   maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="DateTime" type="TimestampTyp"/>
          <xs:element name="StateArriveLeave"
                      type="StateArriveLeaveTyp"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="CurrentJobSetTyp">
  <xs:sequence>
    <xs:element name="CurrentJob" minOccurs="0"
                                  maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Project" type="xs:string"/>
          <xs:element name="Description" type="xs:string"/>
          <xs:element name="Job" type="JobTyp"/>
```

```
          <xs:element name="Customer" type="xs:string"/>
          <xs:element name="Article" type="xs:string"/>
          <xs:element name="OperationDescription" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="FavoritJobSetTyp">
  <xs:sequence>
    <xs:element name="Job" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Project" type="xs:string"/>
          <xs:element name="Description" type="xs:string"/>
          <xs:element name="Job" type="JobTyp"/>
          <xs:element name="Customer" type="xs:string"/>
          <xs:element name="Article" type="xs:string"/>
          <xs:element name="OperationDescription" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="BookingHourSetTyp">
  <xs:sequence>
    <xs:element name="BookingHour" minOccurs="0"
                                   maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="PersNo" type="PersNoTyp"/>
          <xs:element name="Duration" type="QuantityDecimalTyp"/>
          <xs:element name="DateLastBooking" type="TimestampTyp"/>
          <xs:element name="Project" type="xs:string"/>
          <xs:element name="Description" type="xs:string"/>
          <xs:element name="Job" type="JobTyp"/>
          <xs:element name="Customer" type="xs:string"/>
          <xs:element name="Article" type="xs:string"/>
          <xs:element name="OperationDescription" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
```

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

<xs:complexType name="ApprovalSetTyp">
  <xs:sequence>
    <xs:element name="Approval" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
          <xs:element name="PersNoEmployee" type="PersNoTyp"/>
          <xs:element name="NameEmployee" type="xs:string"/>
          <xs:element name="Status" type="ApprovalStatusTyp"/>
          <xs:element name="Sort" type="ApprovalSortTyp"/>
          <xs:element name="DateStart" type="TimestampTyp"/>
          <xs:element name="DateEnd" type="TimestampTyp"/>
          <xs:element name="Parameter1" type="xs:string"/>
          <xs:element name="Parameter2" type="xs:int"/>
          <xs:element name="Parameter3" type="xs:int"/>
          <xs:element name="Parameter4" type="xs:int"/>
          <xs:element name="PersNoApprover" type="PersNoTyp"/>
          <xs:element name="NameApprover" type="xs:string"/>
          <xs:element name="PersNoBooker" type="PersNoTyp"/>
          <xs:element name="NameBooker" type="xs:string"/>
          <xs:element name="Comment" type="xs:string"/>
          <xs:element name="DateEmployee" type="TimestampTyp"/>
          <xs:element name="DateApprover" type="TimestampTyp"/>
          <xs:element name="DateBooker" type="TimestampTyp"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ProxyApproverSetTyp">
  <xs:sequence>
    <xs:element name="ProxyApprover" minOccurs="0"
                                     maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="PersNoEmployee" type="PersNoTyp"/>
          <xs:element name="NameEmployee" type="xs:string"/>
          <xs:element name="Fixed" type="YesNoTyp"/>
```

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

<xs:complexType name="AbsenceSetTyp">
  <xs:sequence>
    <xs:element name="Absence" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="PersNo" type="PersNoTyp"/>
          <xs:element name="Date" type="TimestampTyp"/>
          <xs:element name="AbsenceSort" type="AbsenceSortTyp"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ApprovalBookingDataSetTyp">
  <xs:sequence>
    <xs:element name="ApprovalBookingData" minOccurs="0"
                                           maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="ApprovalNo" type="ApprovalNoTyp"/>
          <xs:element name="BookingDataNo" type="BookingDataNoTyp"/>
          <xs:element name="DateStart" type="TimestampTyp"/>
          <xs:element name="DateEnd" type="TimestampTyp"/>
          <xs:element name="Comment" type="TextTyp"/>
          <xs:element name="Job" type="JobTyp"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>



  <!-- Typdefinitionen (Inhalte) -->

<xs:simpleType name="TokenTyp">
```

```
  <xs:restriction base="xs:int">
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="PersNoTyp">
  <xs:restriction base="xs:int">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="999999"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="TimestampTyp">
  <xs:restriction base="xs:double">
   <xs:minInclusive value="631148400"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="StateTyp">
  <xs:restriction base="xs:string">
   <xs:enumeration value="all"/>
   <xs:enumeration value="open"/>
  </xs:restriction>
</xs:simpleType>

 <xs:simpleType name="JobTyp">
  <xs:restriction base="xs:string">
  </xs:restriction>
</xs:simpleType>

 <xs:simpleType name="OperationTyp">
  <xs:restriction base="xs:int">
    <xs:minInclusive value="1"/>
    <xs:maxInclusive value="9999"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="StateBookingTyp">
  <xs:restriction base="xs:string">
   <xs:enumeration value="book" />
   <xs:enumeration value="interrupt" />
  </xs:restriction>
</xs:simpleType>
```

```
 <xs:simpleType name="ApprovalNoTyp">
  <xs:restriction base="xs:int">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="999999999"/>
  </xs:restriction>
</xs:simpleType>

 <xs:simpleType name="TaskTyp">
  <xs:restriction base="xs:string">
   <xs:enumeration value="make"/>
   <xs:enumeration value="erase"/>
    <xs:enumeration value="changepwd"/>
  </xs:restriction>
</xs:simpleType>

   <xs:simpleType name="QuantityTyp">
  <xs:restriction base="xs:int">
    <xs:minInclusive value="-999999999"/>
    <xs:maxInclusive value="999999999"/>
  </xs:restriction>
</xs:simpleType>

     <xs:simpleType name="QuantityDecimalTyp">
  <xs:restriction base="xs:decimal">
    <xs:minInclusive value="-999999999.9"/>
    <xs:maxInclusive value="999999999.9"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="TextTyp">
  <xs:restriction base="xs:string">
    <xs:maxLength value="50"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="SearchTyp">
  <xs:restriction base="xs:string">
    <xs:maxLength value="50"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="LogTaskTyp">
  <xs:restriction base="xs:string">
```

```
   <xs:enumeration value="off"/>
   <xs:enumeration value="internal"/>
    <xs:enumeration value="external"/>
  </xs:restriction>
</xs:simpleType>

 <xs:simpleType name="StateTaskTyp">
  <xs:restriction base="xs:string">
   <xs:enumeration value="open"/>
   <xs:enumeration value="close"/>
  </xs:restriction>
</xs:simpleType>

 <xs:simpleType name="YesNoTyp">
  <xs:restriction base="xs:string">
   <xs:enumeration value="yes"/>
   <xs:enumeration value="no"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="StateArriveLeaveTyp">
  <xs:restriction base="xs:string">
   <xs:enumeration value="arrive"/>
   <xs:enumeration value="leave"/>
   <xs:enumeration value="trip"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="GroupTyp">
  <xs:restriction base="xs:string">
   <xs:enumeration value="all"/>
   <xs:enumeration value="cuser"/>
   <xs:enumeration value="puser"/>
   <xs:enumeration value="dep"/>
  <xs:enumeration value="app"/>
  <xs:enumeration value="pproxy"/>
  </xs:restriction>
</xs:simpleType>


<xs:simpleType name="ApprovalSortTyp">
  <xs:restriction base="xs:string">
   <xs:enumeration value="holiday"/>
```

```
    <xs:enumeration value="overtime"/>
    <xs:enumeration value="arrive"/>
    <xs:enumeration value="leave"/>
    <xs:enumeration value="booking"/>
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="ApprovalStatusTyp">
   <xs:restriction base="xs:string">
    <xs:enumeration value="open"/>
    <xs:enumeration value="reject"/>
    <xs:enumeration value="approve"/>
    <xs:enumeration value="book"/>
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="AbsenceSortTyp">
   <xs:restriction base="xs:int">
     <xs:minInclusive value="0"/>
     <xs:maxInclusive value="999999999"/>
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="BookingDataNoTyp">
   <xs:restriction base="xs:int">
     <xs:minInclusive value="0"/>
     <xs:maxInclusive value="999999999"/>
   </xs:restriction>
</xs:simpleType>

</xs:schema>
```

B.3 Quellkode-Beispiele

B.3.1 Client-Implementierung

```
//Client erstellen und konfigurieren
HttpWebRequest req = (HttpWebRequest)WebRequest
                .Create("https://127.0.0.1:8443/HelloWorld");
req.Timeout = 600000;
req.Proxy = null;
req.Credentials = CredentialCache.DefaultCredentials;

//Anfrage absenden, Antwort empfangen und abspeichern
HttpWebResponse resp = (HttpWebResponse)req.GetResponse();
System.IO.Stream repStream = resp.GetResponseStream();

//Weiterverarbeitung der Antwort
//...

//Stream und Verbindung schließen
repStream.Close();
repStream.Dispose();
req.Close();
```

B.3.2 Server-Implementierung

```
//Server erstellen, konfigurieren und starten
HttpListener listener = new HttpListener();
listener.Prefixes.Add("https://127.0.0.1:8443/");
listener.Start();

//Anfrage erwarten, empfangen und abspeichern
HttpListenerContext context = listener.GetContext();
HttpListenerRequest request = context.Request;
string requestMessage = System.Web.HttpUtility
                        .UrlDecode(request.RawUrl);

//Verarbeitung der Anfrage
//...

//Antwort erstellen, definieren, umwandeln und absenden
HttpListenerResponse response = context.Response;
string responseMessage = System.Web.HttpUtility
                        .UrlEncode("HalloEcho");
byte[] writeByte = System.Text.Encoding.UTF8
                        .GetBytes(responseMessage);
response.ContentLength64 = writeByte.Length;
response.OutputStream.Write(writeByte, 0, writeByte.Length);

//Übertragung und Server beenden
response.OutputStream.Close();
listener.Stop();
listener.Close();
```

Literaturverzeichnis

[1] Eisenhardt, Martin; Henrich, Andreas; Sieber, Stefanie: Rechner- und Betriebssysteme, Kommunikationssysteme, Verteilte Systeme. - Bamberg : Lehrtext der Universität Bamberg, 2007

[2] Gärtner, Markus: Qualitätssicherung ausgewogen. URL: http://www.heise.de/developer/artikel/Qualitaetssicherung-ausgewogen-1792246.html, verfügbar am 28.01.2013

[3] Ghadir, Phillip; Schirmacher, Philipp: Domain-Driven Design in Clojure. URL: http://www.innoq.com/de/articles/2014/03/ddd-clojure-rating/, verfügbar am 26.03.2014

[4] Heise News: Spezifikation des HTTP/1.1-Protokolls neu aufgeschrieben. URL: http://heise.de/-2217866, verfügbar am 11.06.2014

[5] High Flying: Converting, Casting, Is, As, TryParse, Parse, GetType and typeof. URL: http://www.high-flying.co.uk/c-sharp/converting-casting.html, verfügbar am 21.06.2014

[6] ISO/IEC 27000 - Information security management systems; Overview and vocabulary

[7] ITU-T X.200; Information technology - Open Systems Interconnection - Basic Reference Model: The basic model

[8] ITWissen: UDP (user datagram protocol). URL: http://www.itwissen.info/definition/lexikon/user-datagram-protocol-UDP-UDP-Protokoll.html, verfügbar am 03.06.2014

[9] Microsoft: Makecert.exe (Certificate Creation-Tool). URL: http://msdn.microsoft.com/de-de/library/bfsktky3%28v=vs.110%29.aspx, verfügbar am 26.06.2014

[10] Microsoft: SQL Server Profiler. URL: http://msdn.microsoft.com/de-de/library/ms181091.aspx, verfügbar am 26.06.2014

[11] Microsoft: Using Netsh. URL: http://technet.microsoft.com/de-de/library/bb490939%28en-us%29.aspx, verfügbar am 26.06.2014

[12] Microsoft: Verwenden von SQL Server Management Studio. URL: http://msdn.microsoft.com/de-de/library/ms174173.aspx, verfügbar am 26.06.2014

[13] Möller, Thor; Dörrenberg, Florian: Projektmanagement. - 1.Aufl. - Oldenbourg : Oldenbourg Verlag, 2003

[14] Quade, Jürgen; Kunst, Eva-Katharina: Linux-Treiber entwickeln, - 3.Aufl. - Heidelberg : dpunkt.verlag, 2011

[15] Rebenich, Till: Vielgescholten und selten genutzt: Domain-Driven Design und modellgetriebene Softwareentwicklung. URL: http://flurfunk.sdx-ag.de/2013/02/vielgescholten-und-selten-genutzt.html, verfügbar am 06.03.2013

[16] Schäfer, Werner: Softwareentwicklung. - 1.Aufl. - München : Pearson Education Deutschland GmbH, 2009

[17] Schnabel, Patrick: SSL - Secure Socket Layer. URL: http://www.elektronik-kompendium.de/sites/net/0902281.htm, verfügbar am 03.06.2014

[18] Spinczyk, Olaf: Betriebssystem (BS) - Multiprozessorsysteme. - Dortmund : Lehrtext der TU Dortmund, 2010

[19] Universität Wuppertal: Diffie-Hellman-Algorithmus. URL: http://ddi.uni-wuppertal.de/material/spioncamp/dl/austausch-diffie-hellman-station2.pdf, verfügbar am 20.06.2014

[20] Werner, Matthias: Fehler in Software. URL: http://osg.informatik.tu-chemnitz.de/lehre/old/ss08/ds/09-softwareft-a4.pdf, verfügbar am 20.07.2014

[21] Wikipedia. URL: https://de.wikipedia.org, verfügbar am 10.05.2014

[22] Winkler, Peter: Computerlexikon 2010. - 1.Aufl. - München : Pearson Education Deutschland GmbH, 2009

[23] Wireshark: User's Guide. URL: http://www.wireshark.org/docs/wsug_html_chunked/, verfügbar am 26.06.2014

[24] Zehnder, Carl August: Informationssysteme und Datenbanken. - 6.Aufl. - Zürich : vdf-Hochschulverlag AG, 1998

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 01.08.2014