MASTER THESIS

zur Erlangung des akademischen Grades
„Master of Science in Engineering“
im Studiengang Multimedia uns Softwareentwicklung

Design und Implementierung einer Multi-Touch optimierten Android Runtime Umgebung für JVx ERP Applikationen auf Smartphone Devices

Ausgeführt von: Michael Hofer, BSc
Personenkennzeichen: 1210299015

1. Begutachter: DI Dr. Markus Schordan
2. Begutachter: Roland Hörmann

Wien, 22.5.2014

FACHHOCHSCHULE
TECHNIKUM WIEN

Eidesstattliche Erklärung

„Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Ich versichere, dass die abgegebene Version jener im Uploadtool entspricht."

Ort, Datum

Unterschrift

Kurzfassung

JVx ist ein quelloffenes, auf Java basierendes Applikations-Framework.
Es bildet eine Basis um performante und individuell anpassbare ERP Lösungen zu entwickeln.

Der Boom von mobilen Endgeräten wie Smartphones oder Tablets eröffnet neue Perspektiven. Auch für Firmen werden solche Geräte immer interessanter und werden als mobiler Wegbegleiter unumgänglich. So kann beispielsweise von überall aus, zu jeder Tageszeit auf wichtige Daten zugegriffen werden.

Um JVx für diesen Trend fit zu machen, wurde in dieser Masterarbeit ein Konzept ausgearbeitet um JVx-basierende ERP-Systeme auf Smartphones abbilden zu können. Die Umsetzung erfolgte anhand einer Multi-Touch optimierten Android Runtime Umgebung.

Die Herausforderung dabei liegt einerseits bei der Analyse komplexer Datenstrukturen und andererseits bei deren Aufbereitung. Damit soll eine benutzerfreundliche Darstellung auf mobilen Endgeräten gewährleistet werden.

Die Android App wurde als “universelle” Anwendung entwickelt - vergleichbar mit einem Terminal. Sie kann dank eines einheitlichen Kommunikationsprotokolls und einer generischen Präsentationsschicht ohne Anpassungen am Client-Code mit jeder beliebigen JVx Anwendung verbunden werden und deren Inhalte darstellen.

Schlagwörter: JVx, ERP, Framework, Android App, RESTful Service

Abstract

JVx is an open source, java-based application framework that enables the creation of powerful and customizable ERP solutions.

The boom of mobile devices creates new possibilities on the market which companies use for their daily business, namely to access important documents from everywhere at anytime.

To make JVx fit for this trend a proof of concept was created to model JVx-based applications on smartphones. Therefore a multi-touch optimized Android runtime was developed.
The challenge is to analyze complex data structures and to optimize these data for display on mobile devices, an important usability aspect.

The Android application represents a terminal-like solution that connects to each JVx-based ERP Application without changes to the source code on the client side. The standardized communication protocol and the generic presentation layer make it happen.

Keywords: JVx, ERP, Framework, Android App, RESTful Service

Danksagung

Die Firma SIB Visions hat es ermöglicht im Rahmen ihrer Produktweiterentwicklung ein spannendes Projekt für meine Master Thesis anzubieten. Das ganze Team hat mich dabei tatkräftig unterstützt. Vor allem geht mein Dank an Renè Jahn, der mich bei der Umsetzung unterstützte. Danke für das Engagement und die gute Zusammenarbeit.

Ein besonderer Dank geht an meine beiden Betreuer Herrn DI Dr. Markus Schordan und Herrn Roland Hörmann die mir konstruktives Feedback lieferten und mir mit Rat und Tat zur Seite standen.

Last but not least möchte ich mich bei meiner Familie bedanken, vor allem bei meiner Partnerin und zukünftigen Frau Ines, für die Unterstützung und Geduld während meines Studiums.

Inhaltsverzeichnis

1 Einleitung

In diesem Abschnitt werden der Aufbau und der Kontext dieser Arbeit beleuchtet sowie die Motivation und Herausforderungen beschrieben.

1.1 Motivation

Seit der Kommerzialisierung des Internets 1987 und dessen zunehmender Popularität entstanden neue Geschäftsmodelle. Begriffe wie „Online-Banking“, „Online-Shop“ oder „Internet-Telefonie“ boomten. Unternehmen wie Google, Amazon, eBay & Co entstanden und baten ihre Dienste an.
Um dem großen Andrang standzuhalten mussten interne Prozesse optimiert und teilweise automatisiert werden. Sei es bei der Abwicklung von Zahlungen, dem Verwalten von Kundendaten, Lagerbestände oder sonstigen Ressourcen wie Personalverfügbarkeit.

Das digitale Datenaufkommen vermehrte sich. Es mussten Lösungen geschaffen werden um die Informationsvielfalt zu erfassen und gewinnbringend auszuwerten.
Der Markt reagierte indem er entsprechende Lösungen für die Planung einzelner Ressourcen zur Verfügung stellte. Durch die Online-Integration[1] wurde eine standortunabhängige Planung und Kommunikation ermöglicht.

Immer mehr Unternehmen nutzen diese Weiterentwicklung um interne Geschäftsprozesse, wie beispielsweise den Einkauf und Verkauf sowie Kundenangelegenheiten zu optimieren. Sogenannte ERP Systeme nahmen Vormarsch.
Dadurch sind Daten und Informationen jederzeit und von überall aus abrufbar.
Dies führt zu Automatisierungen in einzelnen Geschäftsbereichen und in weiterer Folge zu Zeit- und Kostenersparnissen.

Dieses hohe Innovationstempo machte sich in der gesamten Informations- und Kommunikationstechnik bemerkbar. [1]

Um die anwachsenden Datenmengen verarbeiten und übermitteln zu können wurden neue Standards entwickelt. Schnellere Internetleitungen und kabellose Technologien wurden für die Datenübertragungen eingesetzt.
Der Trend zu mobilen Wegbegleitern wie z.B. Smartphones oder Tablets wurde bemerkbar.
Dieser Aspekt „Mobilität im Geschäftsprozess“ wird im Rahmen dieser Master Thesis – im Speziellen in der Forschungsfrage behandelt.

[1] Zugänglichmachen von unternehmensinternen Informationssystemen über das Internet

Mobile Devices haben einige Einschränkungen wie z.B. kleinere Bildschirme. Auch die Interaktionsmuster sind auf mobilen Geräten anders als auf Desktoprechnern.
Die Herausforderung dieser Arbeit ist es Informationen für diese Gegebenheiten so aufzubereiten, dass sie eine benutzerfreundliche Darstellung auf mobilen Endgeräten ermöglicht.

Die Motivation begründet sich durch eine zunehmende Integration mobiler Mitarbeiter in den Geschäftsprozess. Durch das Aufkommen von Smartphones und Tablets bilden diese die ideale Basis.
Dank einer Anbindung an interne Informationssysteme können somit Daten optimal bereitgestellt werden. Im Kapitel 2.4 „Mobilität in Geschäftsprozessen“ wird darauf näher eingegangen.

1.2 Forschungsfrage

Die Forschungsfrage

„*Welchen Stellenwert hat die Mobilität im Unternehmen und wie kann sie in Geschäftsprozesse integriert werden?*“

beschäftigt sich mit den Problematiken mobiler Mitarbeiter und beleuchtet das Potenzial für Unternehmen, welches im Zuge einer Integration in den Geschäftsprozess entsteht.

1.3 Relevanz und Bedeutung der Arbeit

Open Source Systeme haben einen großen Nachteil - Der „Support“.
Große ERP Konzerne können gezielt an die Wünsche des jeweiligen Unternehmens angepasste Systemadaptionen durchführen sowie gezielte Schulungen und Trainings anbieten.

Ebenso mit im Portfolio befinden sich diverse Versionen des Produkts und sind somit auf den gängigen Plattformen lauffähig. Es werden von Webanwendungen über Desktopanwendungen bis hin zu mobilen Devices, wie z.B. Smartphones oder Tablets alle Bereiche abgedeckt. Native Apps stehen zumindest für das von Google entwickelte Betriebssystem „Android“ sowie für das von Apple entwickelte Betriebssystem „iOS“ zur Verfügung.

Wie Sie im Kapitel „Open Source ERP Lösungen“ lesen werden steht hinter populären OSS Projekten eine große Community. Trotz dieser ist es nicht möglich den Support mit Marktführern wie z.B. SAP zu vergleichen und kann mitunter sehr kostenintensiv sein.

Ein „All-In-One“ Paket im OSS Bereich welches alle gängigen Plattformen nativ unterstützt – sei es im Web-, Desktop- oder Mobile-Bereich - ist nicht zu finden.
Die Firma SIB Visions hat ein sehr flexibles anpassbares ERP Framework entwickelt welches als Grundlage für diese Master Arbeit dient um diese Lücke zu schließen.

Da ein Framework eine Basis für Entwickler darstellt und ohne speziellen Programmierkenntnissen nicht verwendbar ist, existiert auch eine grafische Oberfläche für das Zusammenstellen der eigenen ERP Anwendung, die sich auf verschiedene Plattformen exportieren lässt.

Um das JVx fit für den mobilen Einsatz zu machen wurde im Rahmen der Arbeit eine mobile Kommunikationsschnittstelle entworfen. Diese dient als Grundlage für die Entwicklung mobiler Anwendungen.

Das Hauptziel dieser Arbeit ist es eine native Androidanwendung zu schreiben um auf JVx-basierende Informationssysteme zugreifen zu können.
Das Userinterface soll generisch entwickelt werden. Damit ist es möglich eine Anwendung ohne Anpassungen am Code für jede beliebige in JVx entwickelte ERP-Anwendung verwenden zu können.

Um dies bewerkstelligen zu können musste ein Konzept erstellt werden um große Datenmengen so aufzubereiten, dass sie an Limitationen mobiler Endgeräte angepasst sind.
Native Interaktionsmuster der Plattform Android wurden analysiert und so kombiniert, dass ein Navigieren durch eine große Anzahl von (verschachtelten) Tabellen einfach möglich ist.

Eine native Anwendung für iOS Geräte rundet das Angebot ab. Diese ist nicht Teil dieser Arbeit. Sie wurde von meinem Studienkollegen Herrn Fessler entwickelt. Seine Erkenntnisse und Ergebnisse sind in seiner Master Thesis „Design und Implementierung einer Multi-Touch optimierten nativen iOS App für das JVx ERP Applikation Framework“ ersichtlich.

Da die Integration mobiler Geräte immer mehr Einzug in den Geschäftsalltag findet und unterschiedlichen Anforderungen gerecht werden muss, ist der Autor dieser Arbeit der Meinung, dass durch die Entwicklung einer generischen, automatisch anpassbaren Anwendung eine Lösung geschaffen wurde um auf den Zug der Mobilität aufspringen zu können. Durch einen hohen Grad an Skalierbarkeit ist die Anwendung für zukünftige Anforderungen gerüstet.

1.4 Aufbau der Arbeit

Die Master-Thesis umfasst 7 Kapitel.

Um einen Überblick über die Grundlagen von ERP-Systemen zu bekommen, werden im Kapitel „ERP Systeme – Ein Überblick**“** neben der technischen Sicht auch grundlegende Begriffe sowie wirtschaftliche Faktoren erläutert. Ebenso wird der Stellenwert und „State of the Art“ von ERP Systemen analysiert.

Die Integration der Mobilität in Geschäftsprozesse bietet weitere Vorteile für Unternehmen. Diese, sowie die Akzeptanz werden anhand einer Literaturrecherche bewertet. Sie bilden die Grundlage für die Beantwortung der Forschungsfrage und werden im Kapitel 2.4 behandelt.

Ein Ziel dieser Arbeit ist es eine technologie- und plattformunabhängige Kommunikationseinheit zu entwickeln (JVx.mobile), welche als Basis für den Entwurf des mobilen Android Clients dient. Das Kapitel „Architektur von ERP Systemen“ beschreibt verschiedene Konzepte und Architekturen und bildet die theoretische Basis für den Entwurf eines Servers, der für die Kommunikation zwischen Client und JVx dienen soll.
Das von SIB Visions entwickelte ERP Framework wird ebenfalls besprochen. Dabei werden Unterschiede, sowie Vor- und Nachteile von Frameworks erläutert.

Um eine technologie- und plattformunabhängige Kommunikationsschnittstelle über das Internet zwischen mobilen Geräten und JVx bereitzustellen, ist der Einsatz eines Webservices unumgänglich. Das Kapitel „JVx.mobile“ stellt standardisierte Konzepte gegenüber und erläutert Designentscheidungen, welche für die Entwicklung des Servers herangezogen wurden.

Das primäre Ziel dieser Arbeit ist das Design sowie die Implementierung einer Multi-Touch optimierten Android Runtime Umgebung. Im Kapitel „MyERP - Mobiler Android Client“ werden die dafür benötigten Konzepte und Interaktionsmuster erläutert. Anhand der für diese Master Thesis entwickelten Beispielanwendung “MyERP” werden Designentscheidungen erläutert um komplexe Datenstrukturen auf kleinen, mobilen Displays abbilden zu können.

Im Kapitel 6 wird auf die Usability des Android Client eingegangen und Standardkonzepte diskutiert.

Das letzte Kapitel dient als Zusammenfassung und Darstellung der wichtigsten Erkenntnisse und bietet einen Ausblick in zukünftige Entwicklungen.

2 ERP Systeme – Ein Überblick

Dieses Kapitel liefert Grundlagen um die Wirtschaftlichkeit von ERP Systemen besser zu verstehen und bildet die Basis für die Beantwortung der Forschungsfrage.

ERP ist die Abkürzung für „Enterprise Resource Planning“. ERP Systeme verknüpfen Informationen und Daten verschiedenster Unternehmensabteilungen um Geschäftsprozesse sowie die Planung in Unternehmen zu optimieren.

Schon seit Beginn der Industriellen Revolution wurden Systeme für die Planung und Kontrolle der Lagerbestände entwickelt. So genannte „Reorder Point (ROP) Systeme“ entstanden. Sobald ein Lagerbestand eine definierte kritische Menge erreicht, konnte das System aufgrund historischer Daten den zukünftigen Bedarf ermitteln. [2]

1970 entwickelte sich dieses Konzept zu einer computerbasierten, automatisierten Materialbedarfsplanung (MRP, engl. Materials Requirement Planning). Das Ziel war die Optimierung der Materialbeschaffung durch Erhebung stochastischer Werte.
Durch die Integration neuer Funktionen für den Materialeinkauf sowie für die Termin- und Kapazitätsplanung entstand 1980 der Begriff „MRP II“ (engl. Manufacturing Resource Planning). Somit war es möglich den Materialbeschaffungsprozess mit dem Produktionsprozess zu verknüpfen. [3]

Auf Grund des hohen technischen Fortschritts in den späten 90er Jahre wurden Organisationseinheiten umstrukturiert um dem Wettbewerb stand zu halten. Vorhandene Informationen mussten besser verarbeitet und verwertet werden um schneller auf Änderungen des Marktes reagieren zu können.
Abteilungs- und unternehmensübergreifende Planungen waren notwendig um den Absatz sowie Umsatz zu steigern.

Der Begriff der „Integration“ gewann an Bedeutung. Er ist wegweisend für heutige ERP-Systeme.
Während ROP und MRP Systeme vertikal in einzelne Teile des Geschäftsprozesses integriert waren, greifen ERP-Systeme horizontal in die komplette Wertschöpfungskette ein. Sie vereint die Bereiche Marketing und Vertrieb, Produktmanagement, Abrechnung und Finanzen bis hin zum Personalwesen (HRM, engl. Human Resource Management). [3]
Durch Schnittstellen können ERP Systeme eingesetzt werden um Geschäftsprozesse mit Kunden und Lieferanten unternehmensübergreifend zu koordinieren. [4]

2.1 Stellenwert von ERP Systemen

Wie im vorigen Kapitel beschrieben werden ERP Systeme eingesetzt um interne sowie unternehmensübergreifende Prozesse zu unterstützen. Dies ist möglich durch eine gemeinsam genutzte Datenbasis.
Daraus ergibt sich ein großer Wettbewerbsvorteil gegenüber Konkurrenten. Informationen können gezielt ausgewertet werden und gelangen schneller an den entsprechenden Personenkreis. [4]

Folgende Auflistung zeigt die Top 10 Erfolgsfaktoren, die ein Unternehmen durch den Einsatz von ERP Systemen erzielen kann. Die nachfolgende Liste basiert auf einer Studie von Accenture[2] die 2006 durchgeführt wurde. [5]

1. Qualitative Steigerung von Management-Entscheidungen
2. Vorteile im Finanzmanagement
3. Schnellere und zeitnahe Transaktionen werden ermöglicht
4. Kostenreduktion
5. Einfache Erweiterbarkeit und hohe Flexibilität
6. Reduzierung der Personalkosten
7. Verbesserungen im CRM Bereich
8. Wettbewerbsvorteil
9. Reduzierung von physikalischen Ressourcen (z.B. LKWs, Lagerhallen,..) sowie Verbesserung der Logistik
10. Steigerung des Umsatzes

Gronau [3] nennt noch zwei weitere Vorteile von ERP Systemen.
Zum einen wäre dies die „Automatisierung von Abläufen“ und zum anderen die „Standardisierung von Prozessen“.

Standardisierung bringt durchaus Vorteile wie die Erhöhung der Produktivität, erleichtert die Koordination oder entlastet beispielsweise Führungskräfte durch einen Automatisierungsanteil.
Auf der anderen Seite sagt Gronau [3], dass ein zu hoher Grad an Standardisierung einige Nachteile mit sich bringt. Die Flexibilität leidet.
Individuelle Anpassungen werden mit zunehmenden Grad erschwert. Dies macht sich in hohen Aufwänden und folglich hohen Kosten bemerkbar.

[2] Institute for High Performance Business

2.1.1 Chancen gegenüber Individualsoftware

Neben fertigen ERP Lösungen gibt es auch die Möglichkeit individuell an das Unternehmen zugeschnittene Informationssysteme entwickeln zu lassen.
Solch eine Entscheidung kann sich als langwierig und kostenintensiv darstellen. Die Flexibilität und Erweiterbarkeit jener Lösungen ist viel geringer als bei „Out-Of-The-Box“ Lösungen. [6]

Im nächsten Abschnitt Big Players am ERP Markt wird eine Übersicht über die marktführenden Unternehmen gegeben.

2.2 Big Players am ERP Markt

Der Markt in diesem Segment ist sehr unübersichtlich.
Da ERP-Systeme ein breites Spektrum an Geschäftsprozessen abbilden müssen, gibt es eine Vielzahl an Anbietern.

Tabelle 2-1 zeigt verschiedene Differenzierungsmerkmale:

Funktionsumfang	Materialplanung, Leistungserstellungen, Personalmanagement, Finanzmanagement, Vertrieb
Grad der Spezialisierung in der jeweiligen Branche	Anlagenbau, Fleischverarbeitung oder beispielsweise in der Pharmaindustrie
Zahl der Nutzer/Anwender	Kleinbetriebe bis hin zu Konzernlösungen
Regionale Verbreitung	Länderspezifische Funktionen oder einfacherer Support durch Hersteller

Tabelle 2-1: Differenzierungsmerkmale von ERP Systemen [3]

Die Top 3 Firmen am ERP Markt sind „SAP“, „Oracle“ und „Microsoft“. Diese Unternehmen bieten branchenübergreifende Lösungen – für Kleinunternehmen bis hin zu großen Konzernen. [7]
Alle erwähnten Unternehmen entwickeln auch mobile Lösungen. Damit ist es möglich, unterwegs zu jeder Zeit in Echtzeit auf Informationen zuzugreifen. Der Prozess der Informationsbeschaffung wird dadurch optimiert.

SAP ist der weltweit marktführende ERP Konzern. In Tabelle 2-2 wird der Funktionsumfang von SAP Lösungen gelistet. Dieser ähnelt denen der Konkurrenten.

	Self Services					
Analytics	Strategic Enterprise Management	Financial Analytics	Operations Analytics	Workforce Analytics		
Financials	Corporate Governance	Financial Accounting	Management Accounting	Financial Supply Chain Management		
Human Capital Management	Employee Relationship Management	Employee Lifecycle Management	Employee Transaction	Worforce Deployment		
Operations: Wertschöpfung	Einkauf	Bestands-führung	Produktion	Distribution	Kunden-auftrags-abwicklung	Service-auftrags-abwicklung
Operations: Support	Produktstruktur-management	Projekt-management	Qualitäts-management	Anlagen-management		
Corporate Services	Travel Management	Environment, Health & Safety	Provisions-management	Immobilien-management		
Lösungs- und Integrations-plattform	Integration von Personen	Integration von Informationen	Integration von Prozessen	Anwendungs-plattform		

Tabelle 2-2: Funktionsumfang von SAP ERP und Integration einzelner Bereiche [8]

2.3 Open Source ERP Lösungen

Definition Open Source

Bonaccorsi [9] beschreibt den Begriff Open Source (Quelloffen) als einen innovativen Prozess um uneingeschränkten Zugriff auf den Quellcode zu gewähren.
Dieser darf im Rahmen der mitgelieferten Lizenz frei verwendet werden.
Die beinhaltete Lizenz regelt alle Bedingungen die erfüllt sein müssen um beispielsweise Modifikationen oder Adaptionen vornehmen zu dürfen.

Die Open Source Initiative (OSI) ist der Meinung, dass es viel mehr ist als das Bereitstellen von Source Code. Dazu definiert sie verschiedene Kriterien, die für eine Open Source Lizenz erfüllt sein müssen. In [10] werden diese beschrieben.

[11] [12] definiert 3 Hauptkategorien von Open Source Software (OSS, engl. Open source software):

Software mit Copyleft
Zu der am meist verbreitetsten Lizenz dieser Kategorie gehört die GNU General Public License (GNU GPL). Der Nutzer, die Nutzerin dieser Lizenz verpflichtet sich bei Verbreitung oder Erstellung von Derivaten das Ergebnis ebenfalls unter die gleiche Lizenz zu stellen.
Das Zusammenführen von Software unterschiedlicher Lizenzen ist hier ebenfalls verboten. Dies verhindert, dass der Source Code oder Teile davon für proprietäre Software verwendet wird.

Software ohne Copyleft
Anders als bei Software mit Copyleft ist es möglich Softwareprodukte oder Derivate, die mit einer Lizenz dieser Kategorie versehen sind, unter anderen Lizenzbedingungen zu vertreiben.
Vertreter dieser Kategorie sind z.B. Berkeley Software Distribution License (BSD License) oder die Apache License.

Lizenzen mit eingeschränktem Copyleft
Diese Kategorie stellt eine Mischform dar. Ein Vertreter ist z.B. Mozilla Public License (MPL)

Laut der Berner Konvention unterliegen alle Niederschriften automatisch dem Urheberrecht. Dies ist auch auf Software anzuwenden. Es stellt hohen bürokratischen Aufwand dar, wenn Software in die Gemeinfreiheit „Public Domain“ entlassen werden soll.

Es müssen rechtliche Schritte eingeleitet werden um auf das Copyright lt. Urheberrecht zu verzichten. [11]

Die Community von OSS nimmt stetig zu. Der Trend ist auch im Bereich quelloffenen ERP Systemen zu merken. Da Informationsmanagementsysteme nicht „plug-and-play“ fähig sind, müssen bei der Einführung entsprechende Adaptionen der Software miteingeplant werden.

In [14] werden folgende Vorteile von OSS gegenüber kommerziellen Lösungen aufgelistet:

- **Hohe Anpassbarkeit**: Durch den vollen Zugriff auf den Source-Code kann das System einfacher an Geschäftsprozesse angepasst werden.
- **Geringere Abhängigkeit** zu einem einzigen Hersteller: Wenn Anbieter vom Markt verschwinden wird es bei proprietären ERP Lösungen schwer bis unmöglich ohne Support weiterzupflegen. Dieses Risiko wird bei OSS abgemildert.
- **Kostenersparnis**: Lizenzen entfallen im Gegensatz zu SAP, Oracle oder anderen kommerziellen Lösungen. Zusätzlich wird keine spezielle Hardware benötigt um quelloffene ERP Systeme zu installieren und zu starten.

[15] vergleicht in einer empirischen Studie den Aufbau und Struktur von Open Source und proprietären Code.
Daraus geht hervor, dass OSS modularer aufgebaut ist. Den Grund dafür gab MacCormack [15] der hohen Anzahl von unterschiedlichen Entwicklern die an einem Projekt beteiligt sind. Um gemeinsam erfolgreich ein so großen Projekt umsetzen zu können müssen gewisse Regeln befolgt werden und Programmierstandards eingehalten werden.

Der oben erwähnte Punkt der Unabhängigkeit ist zugleich als Nachteil an OSS zu sehen. Solange der Hersteller am Markt existiert kann davon ausgegangen werden, dass er bei Problemen speziell auf das Produkt geschulte Consultants zur Verfügung stellt oder anderweitig Support anbietet. Dies wird in speziellen Wartungsverträgen (SLA, engl. Service Level Agreement) festgehalten. Zusätzlich werden auch meist Ausbildungs- und Zertifizierungsprogramme geregelt.

Auf Wunsch werden auch maßgeschneiderte Schulungen und Trainings angeboten. Bei OSS kann das in diesem Umfang nicht angeboten werden. [14] Hier kann nur auf den Support der Community zurückgegriffen werden. Diese Tatsache erschwert es die notwendige Unterstützung zu bekommen und ist sehr zeitintensiv. Eine gute Planung und entsprechendes Know-How müssen vorhanden sein um OS Projekte erfolgreich in Unternehmen zu integrieren.

Entscheidungskriterien dafür sind nicht Teil dieser Arbeit. Informationen dazu finden sie im Buch von [3]. Im Kapitel „Auswahl, Einführung und Betrieb von ERP-Systemen“ wird auf diese Themenstellung eingegangen.

Da Unternehmen nicht immer OSS In-House auf die eigenen Bedürfnisse anpassen können, müssen eigene Consultants herangezogen werden. Die Kosten sind nicht zu unterschätzen und können um die Hälfte teurer sein als Consultants von proprietärer Software. [14]

Der Markt bietet viele Möglichkeiten. Um mit einer ERP Einführung den erhofften Erfolg zu erzielen, müssen zusätzlich noch einige andere Faktoren in Betracht gezogen werden. In [3] werden jene Faktoren angeführt und besprochen.

Sourceforge [3] ist die größte Plattform für Open Source Projekte. Johansson und Sudzina [14] analysieren diese Aussage in ihrer Arbeit „ERP systems and open source“ um den Status von Open Source ERP Systemen zu erheben.
Die Aufzeichnung startete im September 2007. Der Markt wurde 10 Wochen lang beobachtet. Innerhalb dieser Zeit wurden 20 neue Open Source ERP Projekte veröffentlich. Das Interesse – gemessen an den Downloadzahlen - an folgenden, angeführten Projekten stieg ebenfalls.

In der Literatur [14] tauchen diese OS ERP Systeme vermehrt auf:

- **Compiere**
  „*Compiere ERP+CRM is the leading open source ERP solution for Distribution, Retail, Manufacturing and Service industries. Compiere automates accounting, supply chain, inventory and sales orders. Compiere ERP is distributed under GPL V2 by Compiere, Inc*.“ [4]

- **Opentaps**
  „*ERP and CRM suite, including eCommerce, inventory, warehouse, order, customer management, general ledger, MRP, POS. Database independent service-oriented architecture (SOA)*“ [5]

[3] http://www.sourceforge.net

[4] http://sourceforge.net/projects/compiere/?source=directory, letzter Zugriff 4.4.2013

[5] http://sourceforge.net/projects/opentaps/?source=directory, letzter Zugriff 4.4.2013

- **WebERP**
  „*Particularly suited to wholesale, distribution, manufacturing and now also as the hub for multi-branch retail businesses (with an external desktop POS add-on)“* [6]

- **ERP5**
  "*ERP5 is a full featured high end Open Source / Libre Software solution published under GPL license and used for mission critical ERP / CRM / MRP / SCM / PDM applications by industrial organizations and government agencies*." [7]

- **OpenBravo**
  „*Openbravo 3, the agile ERP, is a modular, ready to use, 100% web-based open source business management system written in Java, that automates all of the core business processes for small and mid-sized companies.“* [8]

[6] http://sourceforge.net/projects/web-erp/?source=directory, letzter Zugriff 4.4.2013

[7] http://www.erp5.org, letzter Zugriff 4.4.2013

[8] http://sourceforge.net/projects/openbravo/?source=directory, letzter Zugriff 4.4.2013

2.4 Mobilität in Geschäftsprozessen

Dieses Kapitel liefert die Grundlagen um die Forschungsfrage beantworten zu können. Hierfür wurde Literaturrecherche betrieben. Aussagen und Ansätze werden betrachtet sowie der Bedarf an Mobilität in Geschäftsprozessen ermittelt.

IT Einsatz in Unternehmen beschränkt sich schon lange nicht mehr auf operative Abläufe. Es wird vielmehr versucht komplette betriebliche Prozesse abzubilden und zu unterstützen. Sei es auf Grund von Individual- und Branchenlösungen oder durch komplexe ERP Systeme. Angesichts einer zentralen Datenhaltung und der zunehmende Vernetzung über lokale Netzwerke oder sogar über den Internet Stack können diese Informationstechnologien die Wertschöpfungskette eines Unternehmens unterstützen. [16]

Um „Mobilität", bezogen auf den Geschäftsprozess, besser verstehen zu können müssen einige Begrifflichkeiten geklärt werden.

Was ist „Mobilität" eigentlich?
Laut Duden[9] bedeutet Mobilität „Beweglichkeit".
In der Literatur ist folgende Definition zu finden:

„*humans' independency from geographical constraints*" [18]

2.4.1 Dimensionen der Mobilität

Dank dem Aufkommen mobiler Technologien und Services kann durch die Verwendung von Handys oder beispielsweise Tablets eine geographische Unabhängigkeit geschaffen werden. Aufrufen von geschäftsinternen Informationen oder das Anstoßen von internen Prozessen kann Unterwegs oder auf Reisen zu jederzeit erfolgen.
Dies impliziert eine Zeit- sowie eine Kontextunabhängigkeit. [18]

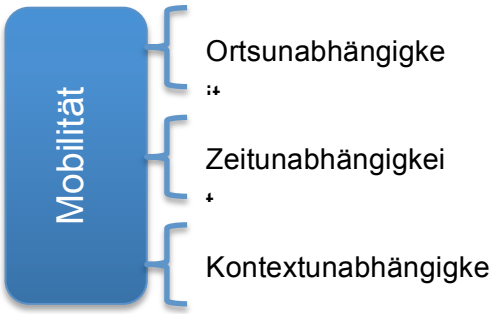

Abbildung 2.1: Dimensionen von Mobilität [18], eigene Überarbeitung

[9] http://www.duden.de

Geografische Dimension

Geografische Dimension oder auch „Ortsunabhängigkeit“ ist verbunden mit Bewegung oder beispielsweise Reisen.
Diese Dimension macht es mobilen Mitarbeitern - Mitarbeiter eines Unternehmens die außerhalb der geschäftlichen Räumlichkeiten tätig sind - möglich mit internen Informationssystemen über das Internet verbunden zu sein. Sie können auf Berichte oder andere Informationen über mobile Endgeräte darauf zugreifen. Informationen können durch einfache Synchronisierung über das Internet aktuell gehalten werden und spart interne Ressourcen für etwaige telefonische Auskünfte. [18]

Zeitliche Dimension

Dank dem Internet sowie neuer Kommunikationstechnologien stehen Daten zu jeder Tageszeit zur Verfügung. Dadurch ist es nicht nur möglich Mitarbeiter auf z.B. anderen Kontinenten mit Daten zu versorgen sondern es werden auch die entstehenden Zeitdifferenzen übergangen. Da Informationen zeitnah bezogen werden können, stellt diese Dimension auch eine Zeitersparnis dar. Es wird der Informationsfluss beschleunigt, sodass geschäftliche Interaktionen zu jeder Ortszeit durchgeführt werden können.
Mitarbeiter haben die Möglichkeit ihre Arbeitsumgebung flexibel zu gestalten. Der Ort sowie die Tageszeit stellen keine Einschränkungen mehr dar. [18]

Kontextuelle Dimension

„Human action is inherently situated in a particular context that frames and is reframed by his or her performance of the action recursively.“ [18]

Der Autor dieses Zitates ist der Meinung, dass die Umgebung in der wir uns befinden unser Handeln beeinflusst. Der Kontext spiegelt sich bei den durchgeführten Aktionen und Interaktionen wider. Eine Aktion ist nur innerhalb des jeweiligen Kontextes nachvollziehbar.

Die Rahmenbedingungen machen es möglich mobile Mitarbeiter, mobile Mitarbeiterinnen in betriebliche Prozesse einzugliedern und mit anderen Akteuren interagieren zu können. In der Literatur wird dafür der Begriff „Mobile Business“ oder auch kurz „M-Business“ verwendet. Eine einheitliche Definition existiert nicht. Eingegliedert wird dieses neue Schlagwort unter den Begriff „Electronic Business“.

Die Schaffung mobiler Arbeitsplätze darf nicht nur aus der technischen Sicht betrachtet werden. Das Unternehmensumfeld sowie die internen Prozesse und Gegebenheiten müssen dafür geschaffen sein. Beispielsweise müssen Informationen von außerhalb des Unternehmens zugänglich sein oder das Senden und Empfangen von E-Mails muss gewährleistet werden.

[16] kategorisiert die Erfordernisse mobiler Technologien nach Tätigkeiten:

- Mitarbeiter, welche sich auf dem Firmengelände an mehreren Standorten aufhalten müssen
- Mitarbeiter, die außerhalb des Firmengeländes ihren Tätigkeiten nachgehen müssen
- Führungskräfte sowie Entscheidungsträger, die operative Tätigkeiten mobil durchführen

Durch stationäre Arbeitsplätze fehlt es an Flexibilität und mobile Mitarbeiter, sowie das Unternehmen an sich, können von den Möglichkeiten, die mobile Technologien mit sich bringen, nicht profitieren.

2.4.2 Eigenschaften der Mobilität

Aus den Merkmalen der Mobilität können Unternehmen Möglichkeiten für den Einsatz mobiler Technologien erarbeiten.

Folgende Eigenschaften werden in [17] zusammengefasst:

Ubiquität und Ortsflexibilität

Darunter versteht der Autor von [17] die permanente Zugriffsmöglichkeit auf unternehmensinterne Informationssysteme unter Verwendung mobiler Informations- und Kommunikationssysteme.

Personalisierung

Ziel der Personalisierung ist das Sammeln, Auswerten und Komprimieren von kundenspezifischen Daten. Hierdurch ist es möglich maßgeschneiderte Informationen dem End-Nutzer aufzubereiten. Dies erleichtert dem Nutzer, der Nutzerin das Selektieren. Auf der anderen Seite hat das Unternehmen den Vorteil gezielt lokale Angebote erstellen zu können. [17]

Verfügbarkeit

Nachdem ein mobiles Zugangsgerät betriebsbereit ist, können ohne Latenzzeiten, Daten und Informationen zeitnah abgerufen werden. Über Datenverbindungen ist es außerdem

möglich, Informationen zu erhalten die nicht explizit angefordert werden. Sofern es der Nutzer, die Nutzerin erlaubt, kann mit der sogenannte „Push-Technik“ Informationen direkt an den End-Nutzer, die End-Nutzerin weitergeleitet werden. Einsatzgebiete wären zum einen Änderungen über Wechsel- oder Aktienkurse. Zum anderen kann eine entsprechende Nachricht gesendet werden, sobald ein Lagerbestand einen Grenzwert erreicht. [17]

Lokalisierungsmöglichkeit

Eine geografische Lokalisierung mobiler Geräte ist ebenso möglich. In Kombination mit der „Personalisierung“ können Informationen und Services anhand des lokalen Kontextes generiert werden. Der Informationsfluss wird dadurch stark optimiert. [17]

Erreichbarkeit

Durch Verwendung mobiler Technologien und Dienste sind Teilnehmer, Teilnehmerinnen jederzeit erreichbar und können auch jederzeit Informationen und Daten erhalten. [17]

2.5 Fazit

Durch den „State of the Art“ im Bereich ERP Systeme und den technologischen Fortschritt im Bereich der Mobilität geht der Autor dieser Arbeit davon aus, dass dieses Thema allmählich in den Geschäftsalltag Einzug findet.

Durch die resultierenden Vorteile sowie Einsatzmöglichkeiten ergeben sich neue Chancen am Markt. Auch für Unternehmen ist der Einsatz von ERP Systemen interessant. Kombiniert mit mobiler Technologie haben Unternehmen einen guten Stand um sich von Konkurrenten abheben zu können.

Wie in diesem Kapitel erwähnt ist noch großes Potenzial vorhanden. Dennoch funktioniert die Integration mobiler Mitarbeiter in den Geschäftsprozess bereits.

Durch die Vorteile mobiler Endgeräten werden die Vorteile der Orts-, Zeit- sowie Kontextunabhängigkeit ausgeschöpft und der beschriebene Mehrwert kann erzielt werden.

3 Architektur von ERP Systemen

In den vorangegangenen Kapiteln wurden wirtschaftliche Aspekte sowie der Stellenwert am Markt behandelt und die Forschungsfrage beantwortet. Dieses Kapitel analysiert technische Anforderungen an ERP Systeme sowie deren Architektur.
Es bildet die technische Grundlage für die Entwicklung der mobilen Kommunikationseinheit „JVx.mobile“.

ERP Systeme bestehen aus mehreren Teilen (Komponenten). Jeder dieser Komponenten hat bestimmte Aufgaben die zu erfüllen sind.
Es müssen Daten verwaltet, sowie anhand bestimmter Geschäftsregeln ausgewertet und erfasst werden können. Um auf diese Daten zugreifen zu können wird noch eine Komponente für die Bedienung einer grafische Oberfläche benötigt. Diese abstrahiert die technischen Funktionen und Abläufe.

Im weiteren Kapitel wird auf diese Anforderungen im Detail eingegangen und Querverbindungen zur Architektur von JVx hergestellt.

3.1 Verteilte Systeme

Eingangs wurde erwähnt, dass ERP Systeme aus mehreren Komponenten bestehen.
Eine Komponente kann beispielsweise ein anderer Computer oder ein Teilprogramm sein, welches eine bestimmte Aufgabe (z.B. das Speichern von Daten oder Informationen) erfüllt. Die einzelnen Komponenten sind so integriert, dass sie vom End-Benutzer, von der End-Benutzerin unbemerkt bleiben.

Aus diesem Sachverhalt heraus wird ein verteiltes System als *„..eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheinen“* [19] definiert.

Dies impliziert, dass die einzelnen Komponenten miteinander kommunizieren müssen sowie diverse Ressourcen (z.B. Hardware und Daten) gemeinsam benutzen.
Dafür ist eine globale Sicht der einzelnen Komponenten auf das Gesamtsystem notwendig. [19]
ERP Systeme folgen diesen Grundsätzen und stellen ein Verteiltes System dar. Im Folgenden werden solche Systeme als „Client-Server Anwendungen“ bezeichnet.
Um die Architektur besser verstehen zu können, werden in den nächsten Unterkapitel die Begriffe Client und Server näher erläutert sowie verschiedene Ausprägungen diskutiert.

Ein **Client** ist ein Programm, welches Dienste eines Servers in Anspruch nimmt. [3]

Er nimmt Benutzereingaben entgegen und leitet sie zur weiteren Verarbeitung an den Server weiter. Nach der Verarbeitung erhält der Client das Resultat seiner Anfrage. [19]
In ERP Systemen stellt der Client eine Benutzerschnittstelle zur Verfügung, welche für die Aufbereitung und der Anzeige von im System erfassten Datensätzen verantwortlich ist (beispielsweise bei der Anzeige der Kundenliste). Um schneller an die gesuchten Daten und Informationen zu gelangen, bietet ein ERP-Client zusätzliche Interaktionsmöglichkeiten an.
Dafür werden neben den Grundfunktionalitäten wie beispielsweise dem Anlegen, Editieren, Löschen von Kundendaten diverse Such- und Filtermöglichkeiten angeboten.
Durch Auslösen eine dieser Aktionen wird eine Transaktion mit dem Server erstellt.

Ein **Server** ist ein Rechner, der in einem Netzwerk bestimmte Aufgaben übernimmt (z.B. das Entgegennehmen von Client-Anfragen sowie die Verwaltung und etwaige Kommunikation mit anderen Komponenten um die angefragten Daten zu erhalten). [3]
Nachdem der Server eine der vorhin erwähnten Anfragen erhält, wird diese entsprechend bearbeitet. Das Resultat ist eine Antwort auf die Anfrage, die dem Client übermittelt wird. Dieser bereitet die Daten auf und gibt sie grafisch aus.

Ein Client läuft meistens auf einem Computer und dient als grafische Schnittstelle zwischen dem Benutzer, der Benutzerin und den erforderlichen Daten welche beispielsweise in einer Datenbank abgelegt sind.
Diese Konstellation wird in der Literatur als „Thin Clients“ bezeichnet.
Der Client kann aber auch die komplette Arbeit des Servers implementieren sowie eine teilweise Persistierung (Datenspeicherung). Diese werden als „Fat Clients“ bezeichnet. [19]
In Abbildung 3.1 werden verschiedene Varianten der Anordnung dargestellt:

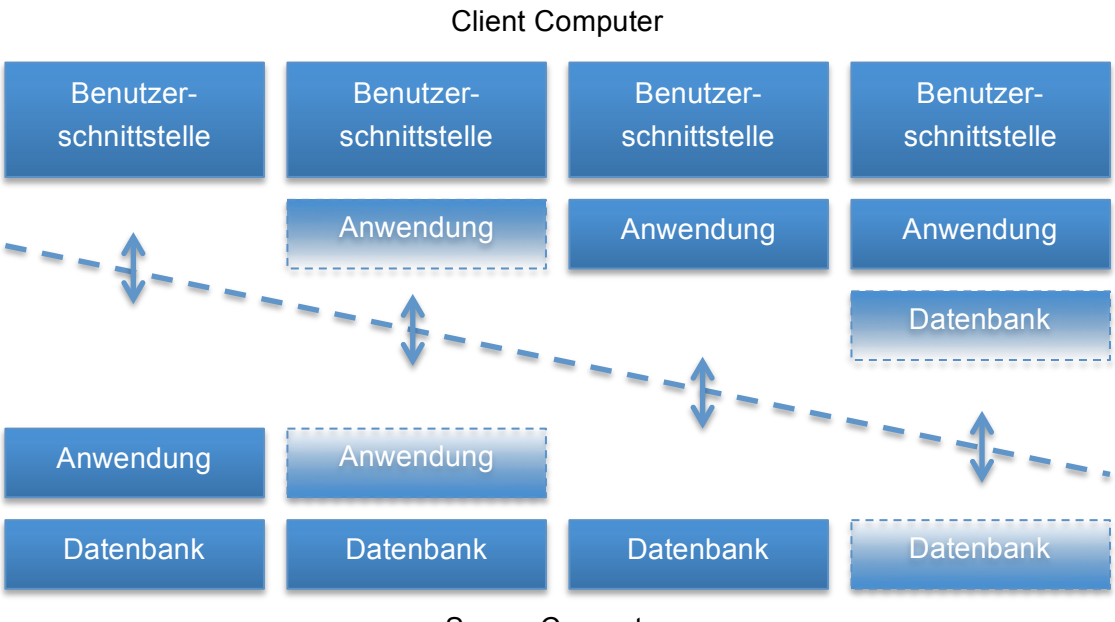

Abbildung 3.1: Anordnung Client-Server Systeme [19], eigene Überarbeitung

3.1.1 Architektur Verteilter Systeme

In Abbildung 3.1 werden die Schichten von Client-Server Architekturen abgebildet.
In dieser Abbildung wird die einfachste Form einer Client-Server Anwendung – eine 2-Schichtenarchitektur - gezeigt. In ERP Systemen fungiert der Server als Client. Die Aufgabe ist es Anfragen vom Client an die Datenschicht weiterzuleiten (siehe Abbildung 3.2). Das erhaltene Ergebnis wird danach geschickt.
Dadurch können die einzelnen Schichten aus Abbildung 3.4 **Error! Reference source not found.**auf unterschiedliche Computer verteilt werden. Dies resultiert in einer 3-Schicht-Architektur (engl. 3-Tier-Architecture), wie in folgender Abbildung dargestellt.

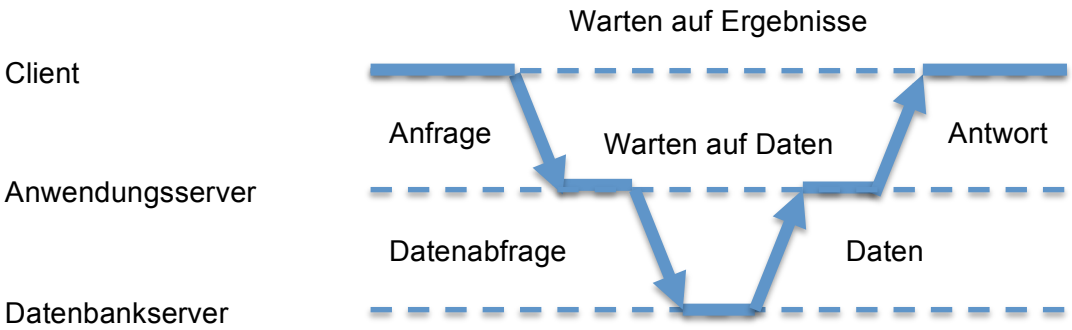

Abbildung 3.2: Server, der als Client fungiert bei synchroner Kommunikation [19], eigene Überarbeitung

Das Java basierte JVx Framework besteht ebenfalls aus drei Tiers, welche in Abbildung 3.4 dargestellt sind. Die Aufgaben und Funktionen der einzelnen Schichten werden anhand der Architektur von JVx im Abschnitt 0 näher erläutert.

Middleware

Zu Beginn des Kapitels wurde erwähnt, dass die Kommunikation eine wichtige Rolle darstellt. Gewünschte Informationen müssen von verschiedenen Datenquellen oder teilweise von anderen Systemen über ein Netzwerk gesammelt werden, damit sie dem Benutzer, der Benutzerin ausgegeben werden können.

Laut Definition (siehe Kapitel 3.1) hat ein verteiltes System den Anspruch seine Komplexität und den Grad der Verteilung zu verbergen. Für den Benutzer, der Benutzerin des Systems ist es weder relevant zu wissen welche Anfragen an welche Komponenten zu stellen sind noch auf welche Art und Weise dies geschieht. Hierfür muss ein Kompromiss gefunden werden zwischen der Transparenz und der Performance eines Systems. (näheres dazu im Kapitel „Architektonische Anforderungen von ERP Systemen“)

Die Middleware stellt für diese Anforderungen die benötigten Dienste (Services) bereit.
Sie bildet eine Schicht zwischen der Anwendung und dem verteilten System.
Des weiteren organisiert sie den Datentransport und Funktionsaufrufe zu anderen im Netzwerk verteilten Systeme und Komponenten. [20]

JVx arbeitet protokollunabhängig. [21]
Durch die abstrakte Implementierung der Transportschicht steht es dem Entwickler, der Entwicklerin offen eigene Protokolle zu implementieren. [22] Auch die Kommunikation des Servers mit der Datenbankschicht ist so aufgebaut, dass eigene Persistenzumgebungen implementiert werden können (siehe Abbildung 3.4).

Kommunikation in verteilten Systemen

Die Kommunikation basiert auf einem Austausch von Nachrichten. Die Koordination dieses Prozesses ist Teil der Middleware.
Bevor die verschiedenen Arten gegenübergestellt werden können, muss der interne Abarbeitungsprozess und die Infrastruktur erklärt werden.
Ein Server in verteilten Systemen muss in der Lage sein mehrere Anfragen gleichzeitig entgegenzunehmen und diese zu bearbeiten. Der Zugriffspunkt wird durch eine Adresse, dem Internet Protokoll (IP) und einem Port definiert. Der Dienst am Server hört diesen ab und nimmt alle Anfragen entgegen. Um eine Parallelverarbeitung zu ermöglichen – sprich das Annehmen von mehreren Anfragen gleichzeitig - wird die Anforderung an einen „Dispatcher“ weitergeleitet. Dieser verteilt sie an diverse Serverprozesse. Dadurch wird die Gesamtleistung optimiert und der Server entlastet. Mehrere eintreffende Anfragen können dadurch zeitgleich bearbeitet werden. [19]

Die Arten der Kommunikation kann man folgendermaßen gliedern:

Asynchrone vs. Synchrone Kommunikation

Die Verwendung asynchroner Kommunikation bringt einige Vorteile mit sich.
Wenn der Sender – bspw. der Client – eine Anfrage an einen entfernten Dienst auf einem Server stellt, wartet er nicht bis er eine Antwort bekommt. Stattdessen können weitere Aktionen die durch den Benutzer, der Benutzerin ausgelöst wurden bearbeitet werden.
Im Gegensatz dazu wartet der Client bei synchroner Kommunikation bis er eine Antwort vom Server bekommt.
Dadurch sind weitere Aktionen clientseitig nicht möglich und werden blockiert oder bis zur Ausführung in einer Warteschlange gespeichert.
Im Einzelfall muss entschieden werden welche Art bevorzug wird. [19]

Persistente vs. Transiente Kommunikation
Bei persistenter Kommunikation werden Nachrichten in der Middleware gespeichert. Sobald die Nachricht ausgeliefert wurde wird der Speicherplatz wieder geleert.
Dies hat den Vorteil, dass die Anwendung nach dem Senden der Nachricht nicht weiter ausgeführt werden muss, da sie durch die Persistierung nicht verloren geht.
Transiente Kommunikation bedeutet hingegen, dass die Nachricht nach beenden der Anwendung ebenfalls aus dem Speicher gelöscht wird. Falls die Nachricht durch etwaige Netzwerk- oder Übertragungsproblemen nicht gesendet werden kann, wird sie ebenfalls verworfen. [19]

In der Praxis finden sich verschiedene Kombinationen wieder. Dadurch können Nachteile der Einzelnen Arten umgangen werden.

In den Kapiteln „JVx.mobile“ und „MyERP - Mobiler Android Client“ wird der in der Master Thesis verwendete Ansatz beschrieben.

3.1.2 Architektonische Anforderungen von ERP Systemen

ERP Systeme müssen eine hohe Anzahl an Benutzern, Benutzerinnenunterstützen. Weiters ist es möglich Anfragen parallel abzuarbeiten und den Zugriff auf Daten zu gewährleisten.
Daraus ergeben sich laut [19] folgende Anforderungen:

- **leichter Zugriff auf Ressourcen**
  Dies stellt das Hauptziel Verteilter Systeme dar.
  Der Benutzer, die Benutzerin muss in der Lage sein auf gemeinsam genutzte Ressourcen (z.B. Drucker, Daten oder Informationen) im Netzwerk zugreifen zu können. Der Informationsfluss sowie die Kollaboration unter den Mitarbeitern und Mitarbeiterinnen wird dadurch optimiert. Bei zunehmender Vernetzung und Kommunikation über Netzwerke (lokal oder über das Internet) spielt das Thema Sicherheit eine große Rolle. Unternehmensrelevante sowie andere sensible Daten (Passwörter oder beispielsweise Zahlungsinformationen) müssen gewissen Sicherheitsrichtlinien entsprechen. Dies sind Herausforderungen, die bei der Planung berücksichtigt werden müssen. [19]

- **Transparenz**
  „Ein verteiltes System, das in der Lage ist, sich Benutzern und Anwendungen so darzustellen, als sei es nur ein einziges Computersystem, wird als transparent bezeichnet.“ [19]

Es gibt verschiedene Formen der Transparenz. Diese können im Buch von [19] nachgelesen werden. Grundsätzlich geht es darum zu verbergen wo sich Ressourcen physisch in einem Netzwerk befinden sowie der Tatsache auf welche Art und Weise darauf zugegriffen wird. Dabei soll es keine Rolle spielen, wie viele konkurrierende Benutzer, Benutzerinnen zeitgleich eine bestimmte Ressource verwenden. Die Maskierung von Fehlern spielt auch eine wichtige Rolle.
Es ist nicht immer sinnvoll eine höchstmögliche Transparenz zu erzielen. Es muss ein Kompromiss geschlossen werden zwischen dem Einsatz von Transparenz und der Performance eines Systems. Dies ist auch für das Verständnis für die Benutzer, Benutzerinnen relevant. [19]

- **Offenheit**
  Offenheit ist ein weiteres wichtiges Ziel. Sie definiert die Kommunikation in verteilten Systemen.

  „Ein offenes verteiltes System bietet Dienste nach Standardregeln an, die die Syntax und die Semantik dieser Dienste beschreiben." [19]

  Regeln werden in Protokollen definiert und steuern das Format, den Inhalt sowie die Bedeutung übermittelter Nachrichten in einem Netzwerk. Die Syntax wird in Schnittstellen (engl. Interfaces) definiert. Diese bilden die Grundlage um eine Kommunikation mit verteilten Systemen zu ermöglichen. [19]

  Des weiteren sollen offene Systeme konfigurierbar und erweiterbar sein. Die Interoperabilität und Portabilität gibt an wie offen ein System entworfen wurde. [19]

- **Skalierbarkeit**
  Ein verteiltes System hat den Anspruch erweiterbar zu sein. Nicht nur in Bezug auf funktionale Anforderungen. So unterscheidet [19] 3 Dimensionen:

  1. **Größe**
     Größenskalierbarkeit ist erforderlich um neue Benutzer sowie beliebige Ressourcen hinzufügen.
     Hier stellt der Server einen Engpass dar, da die Leistung auf seine Hardware-Ressourcen beschränkt ist. Es müssen Strategien entwickelt werden um Last auf mehrere Rechner zu verteilen.
     Das Thema der Dezentralisierung (Dienste, Daten, Algorithmen) spielt hier eine wichtige Rolle um den Server entlasten zu können.
     Durch diese Verteilung entstehen aber auch Sicherheitsrisiken, da mit verschiedenen Rechnern über ein Netzwerk kommuniziert werden muss. Dies muss bei der Entwicklung abgewogen werden. [19]

2. **Geografische Lage**

   Die geografische Skalierbarkeit ist von Problematiken zentralisierter Architekturen abhängig.

   Jede Kommunikation verursacht Latenzzeiten.

   In einem lokalen Netzwerk kann diese relativ gering gehalten werden.

   Bei unternehmens- und länderübergreifenden Systemen sind die Zeiten wesentlich höher.

   Einbußen bezüglich der Leistung und Zuverlässigkeit müssen ebenfalls in Kauf genommen werden. Hier soll die Standortfrage geklärt werden wo und wie Ressourcen ihre optimale Leistung erbringen können. Eine optimale Strategie ist es sie dort zur Verfügung zu stellen wo sie benötigt werden.

   Dies kann durch Replikation von Ressourcen erzielt werden. [19]

3. **Administration**

   Systeme sollen trotz einem hohen Grad an Skalierbarkeit, weiterhin einfach zu administrieren sein. Leistungseinbußen müssen hier in Kauf genommen werden. [19]

3.1.3 Mobilitätsanforderungen für JVx basierte ERP Systeme

Um JVx fit für den Mobile-Hype zu machen muss die vorhandene Architektur erweitert werden. Es wird ein zusätzlicher Server benötigt der in der Lage ist Anfragen von mobilen Android- sowie iOS-basierten Endgeräten entgegenzunehmen.

Durch den objektorientieren, generischen Ansatz von JVx und dessen Implementierung in JAVA ist es möglich vorhandene Objekte und Schnittstellen zu verwenden und das JVx Framework als Grundlage für die Entwicklung eines mobilen, JAVA-basierten Servers (JVx.mobile) heranzuziehen.

Die architektonischen Herausforderungen liegen einerseits bei der Wahl eines geeigneten technologie- und plattformunabhängigem Kommunikationsprotokolls zwischen den mobilen Endgeräten und JVx.mobile und andererseits bei der Transformation der Daten.

Da mobile Endgeräte einige Einschränkungen in Bezug auf die Bildschirmgröße oder die Hardware-Ressourcen im Vergleich zu Desktops mit sich bringen, müssen die Daten am Server entsprechend aufbereitet werden.
Aufgrund dieser Einschränkungen kann nur eine Teilmenge an Daten übermittelt werden um eine möglichst hohe Usability zu erzielen.

Eine weitere Herausforderung besteht darin die übermittelten Datenmengen gering und die Anzahl an Client-Anfragen minimal zu halten. Durch jede Anfrage entsteht eine gewisse Latenzzeit. Bei ständigem Nachladen von Daten ist ein flüssiges Arbeiten am mobilen Gerät nicht möglich.
Diese Designentscheidungen und die resultierende Architektur wird im Kapitel 4 diskutiert.

Auch beim Design sowie bei der Implementierung eines mobilen Clients für ein ERP Framework gilt es einige spezielle Anforderungen zu beachten.
Da JVx höchst generisch aufgebaut ist, muss dies auch für den dafür entwickelten Client gelten um eine Anwendungsunabhängigkeit zu erzielen.
Er wird daher auf Basis eines Thin-Clients entwickelt und stellt eine generische, erweiterbare multi-touch optimierte Anwendung zur Verfügung.
Des weiteren fungiert er als Terminal.

Die aufbereiteten Daten sowie Informationen über die Darstellung und den Seitenaufbau werden vom Server in entsprechend aufbereiteter Form erhalten. Somit ist eine Anbindung an beliebige JVx Applikationen möglich und die Client-Anwendung muss nicht neu programmiert werden.

3.2 ERP vs. Verteilte Systeme - Unterschiede in der Architektur

Ein wesentlicher Unterschied zu verteilten Systemen ist die Flexibilität in Bezug auf die Anpassung an interne Unternehmensstrategien um eine Integration in den Geschäftsprozess zu ermöglichen.

Dafür wird eine eigene Adaptionsschicht benötigt. Sie siedelt sich zwischen der höheren Ebene der Benutzungsschicht und der darunterliegenden Applikationsschicht an. [3] Dies wird in folgender Abbildung verdeutlicht. [3]

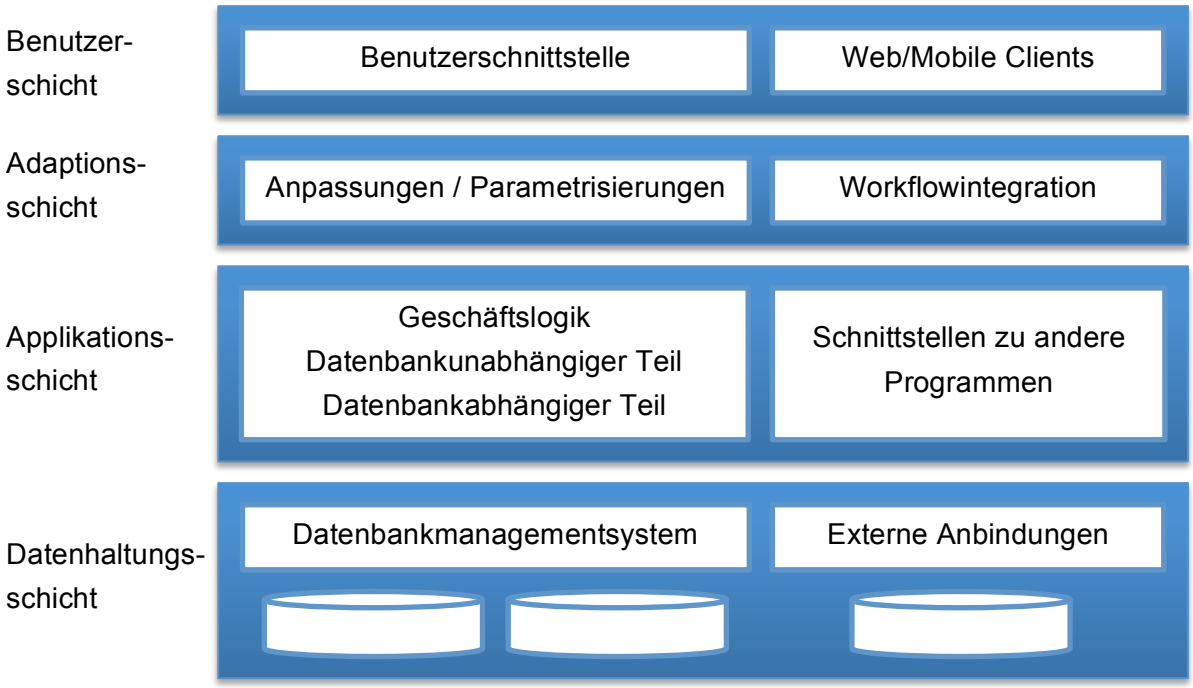

Abbildung 3.3: Architektur eines ERP Systems [3], eigene Überarbeitung

Auf der untersten Schicht befinden sich verschiedenste Informationssysteme. Um auf deren Datenbestand zugreifen zu können werden Datenbankmanagementsysteme (DBMS) eingesetzt. Durch die integrierten Schnittstellen wird eine Kommunikation mit der darauffolgenden Applikationsschicht ermöglicht. [3]

Um unterschiedliche Datenbanksysteme und ihre Standards zu unterstützen wird in dieser Schicht eine Unterteilung in einen datenbankunabhängigen sowie einen datenbankabhängigen Teil vorgenommen. Dadurch können Leistungsoptimierungen

vorgenommen werden. Zusätzlich dient diese Schicht als Ansatz um eigene Erweiterungen zu implementieren. [3]

Um vorhandene Prozesse in die Wertschöpfungskette von Unternehmen integrieren zu können bietet die Adaptionsschicht die notwendigen Einstellungsmöglichkeiten um entsprechende Anpassungen vornehmen zu können. [3]

In der obersten Schicht befindet sich die Ausgabeeinheit. Typischerweise werden heutzutage neben Desktop Anwendungen auch Web sowie Mobile Clients verwendet. [3]

3.3 JVx – Ein open source ERP Framework

JVx ist ein quelloffenes, auf Java basierendes Enterprise Application Framework - entwickelt von der Firma SIB Visions GmbH.
Das Wiener Unternehmen schafft mit ihrem Produkt einen Rahmen um professionelle Datenbank Applikationen entwickeln zu können. Der Entwickler, die Entwicklerin ist in der Lage in kürzester Zeit mit wenigen Code-Zeilen hoch performante Lösungen zu schaffen. Auf der Website[10] werden diese Vorteile durch eine Statistik dargestellt.
Sib Vision hat JVx mit einer Apache Lizenz (Version 2) veröffentlicht.

Der mobile Android Client basiert ebenfalls auf einem vom Autor entwickeltem Framework um eine Applikationsunabhängigkeit zu gewährleisten. In Kapitel 5 wird die Umsetzung des Grundgerüstes erläutert und die Verwendung anhand der Beispielsanwendung „MyERP“ demonstriert.

3.3.1 Definition Framework

Den Begriff Framework definiert Erich Gamma (et. al.) [23] als ein Set von Klassen, die miteinander interagieren. Frameworks sind auf spezielle Anwendungsfälle zugeschnitten und müssen einen breiten Anwendungsbereich abdecken.

Ein Framework stellt nicht den Anspruch vollständig zu sein. Vielmehr handelt es sich um ein wiederverwendbares, generisches Grundgerüst bestehend aus abstrakten Klassen und Richtlinien für die Kommunikation bzw. Interaktion (Schnittstellen, engl. Interfaces), sowie einer Menge von „Hot Spots“. [24]

Aufgrund dieser Rahmenbedingungen können sich Entwickler und Entwicklerinnen auf die Programmierung der eigentlichen Applikation konzentrieren während sich das Framework um die Grundfunktionalitäten kümmert. Beispielsweise übernimmt JVx die technische Implementierung der Datenhaltung oder das Session-Handling.

Software Frameworks helfen den Entwicklungsaufwand und somit Kosten zu reduzieren. Durch das wiederverwendbare Grundgerüst müssen Funktionen, Strukturen sowie Abläufe nicht neu entwickelt werden. Dies steigert die Qualität des Endproduktes. [26]

Abstrakte Klassen

Abstrakte Klassen kapseln Funktionen und Methoden. Sie bilden das Interface für alle abgeleiteten Klassen[11]. Des weiteren können sie ausimplementierte oder abstrakte

[10] www.sibvisions.com

[11] Eine abgeleitete Klasse (Sub-Klasse) stellt eine Spezialisierung einer abstrakten Klasse dar.

Methodenaufrufe - indem sie lediglich die Syntax vorgeben – zur Verfügung stellen. Dessen Implementierung bleibt jeder abgeleiteten Klasse selbst überlassen. Das Verhalten kann ebenso in der Konkretisierung angepasst werden. [23]

Interfaces

Interfaces definieren das Kommunikationsprotokoll zwischen einzelnen Komponenten. Applikationsentwicklern, Applikationsentwicklerinnen wird somit Zugriff auf das Framework gewährt. Interfaces dienen weiter als „Hot Spot“ für eigene Erweiterungen.
Ein Interface gibt an, welche Methoden, Parameter, Variablen verwendet werden und bildet somit die Syntax. Methoden nehmen bestimmte Typen von Parameter entgegen und liefern entsprechende Ergebnisse zurück. Dies wird als Methodensignatur bezeichnet.
In einem objektorientierten System können Objekte nur durch Interface angesprochen werden um mit ihnen in Interaktion zu treten. Sie stellen somit ein wichtiges Programmierkonstrukt dar. [23]

Hot Spot

Ein „Hot Spot“ definiert einen Ansatzpunkt wodurch Entwickler, Entwicklerinnen Zugriff auf das Framework erlangen. Dies können beispielsweise Interfaces sein.

3.3.2 Framework Design

Frameworks weisen einen hohen Grad an Wiederverwendbarkeit auf. Dieses Potenzial kann nur ausgeschöpft werden, wenn der Aufbau und die Struktur verständlich sind um ein effizientes Arbeiten zu ermöglichen. Die zugrundeliegende Komplexität kann dies erschweren.
Um dem Anwendungsentwickler, der Anwendungsentwicklerin den notwendigen Komfort zu bieten muss beim Design darauf geachtet werden, dass Qualitätsmerkmale beachtet und Standards verwendet werden. [25]

Die Domain und der Kontext eines Frameworks müssen spezifiziert werden. Es müssen Einsatzgebiete sowie Verwendungsmöglichkeiten anhand von Beispielszenarien definiert werden. Dies hilft bei der Abstraktion der Anforderungen um eine generische Architektur entwickeln zu können. [25]
Aufgrund der erforderlichen Abstraktion ist die Entwicklung eines Applikations-Frameworks sehr aufwändig und komplex. [26]

Um eine hohe Qualität sicherstellen zu können müssen Standards befolgt werden um den Anforderungen eines Frameworks gerecht zu werden.

Design Patterns

Um Aktionen sowie Abläufe gekapselt in Objekten oder Komponenten wiederverwendbar zu machen ist die Verwendung von „Patterns“ unumgänglich. „Design Patterns“ beschreiben Best-Practices. Sie stellen einen standardisierten Lösungsansatz dar um wiederkehrende Probleme in einem bestimmten Kontext zu lösen.
Für Entwickler, Entwicklerinnen objektorientierter Software bildet dieser Ansatz die Grundlage für die Kommunikation und das Verständnis. Sie sind abstrakter als Frameworks, bilden die kleinste architektonische Einheit und sind die Grundlage für wiederverwendbaren sowie erweiterbaren und verständlichen Code. [27]

Design Patterns finden nicht nur in Frameworks Anwendung sondern bei jedem objektorientierten Software-Design.
In Tabelle 3-1 wird eine Einteilung verschiedener Entwurfsmuster laut [23] gezeigt.

Eine Konkrete Implementierung wird anhand von JVx.mobile im Kapitel 4 sowie bei der Umsetzung der Clientanwendung im Kapitel 5 gezeigt und mit Beispielen erläutert.

Kategorien	Beschreibung
Erzeugungsmuster	Patterns in dieser Kategorie beschäftigen sich mit dem Instanziieren von Objekten. Sie beinhalten Mechanismen um das Erstellen von Objekten optimal zu bewältigen.
Strukturmuster	Strukturmuster beschreiben wie einzelne Objekte und Klassen gekapselt werden können um „Real-World-Szenarien“ abbilden zu können
Verhaltensmuster	Dieses Muster beschäftigt sich mit der Kommunikation sowie Interaktion zwischen Objekten. Hier spielt die Kopplung einzelner Objekte eine wesentliche Rolle. Erzielung einer möglichst losen Kopplung ist auch ein wesentliches Ziel beim Entwurf von Mehrschichtenarchitekturen.

Tabelle 3-1: Arten von Design Patterns [23]

Dokumentation

Neben der Verwendung von Designpatterns spielt die Dokumentation eine wesentliche Rolle. Sie ist sehr aufwendig. Desto wichtiger ist sie für Entwickler, Entwicklerinnen bei der Verwendung und dem Verständnis des Frameworks.

3.3.3 Architektur von JVx

JVx besteht - wie herkömmliche ERP Systeme - aus einer Mehrschichtenarchitektur.
Darunter wird ein Konzept für die Strukturierung von verteilten Systemen verstanden (siehe Kapitel 3.1).
Im Falle von JVx erfolgt diese Verteilung aufgrund von drei Schichten. Sie sind logisch auf unterschiedliche Komponenten verteilt. In der Literatur wird diese Art der Verteilung „vertikale Verteilung“ genannt. [19]
In Abbildung 3.4 wird die Architektur veranschaulicht.

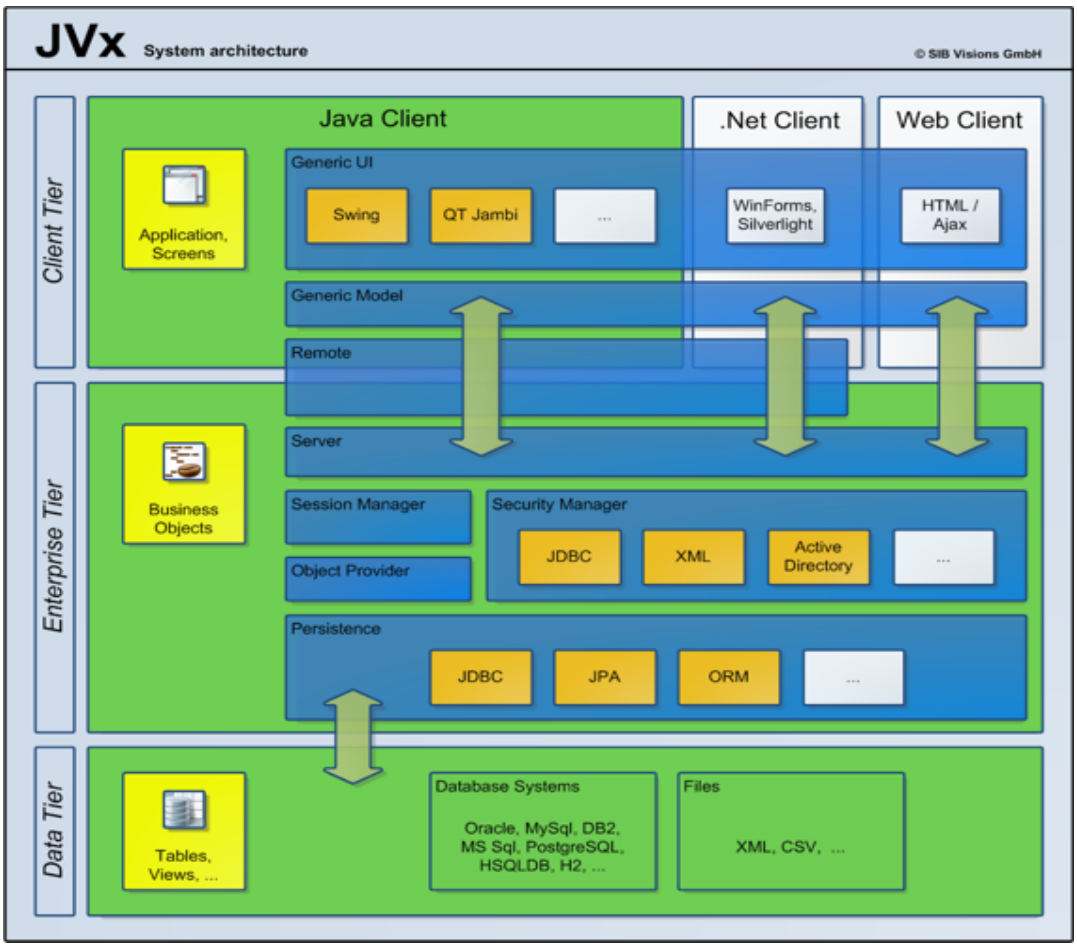

Abbildung 3.4: Architektur von JVx [28]

Die **Datenhaltungsschicht** (Data Tier) unterstützt beliebige relationale Datenbanksysteme sowie dokumentenbasierte Datenformate wie XML oder CSV.

Neben dem Session- und Sicherheitsmanagement ist die **Anwendungsschicht** (Enterprise Tier) für die Verarbeitung der eingehenden Anforderungen zuständig und stellt über eine Schnittstelle (Persistence API) eine Verbindung zur Datenhaltungsschicht her.

Die technologieunabhängige **Darstellungsschicht** (Client Tier) fungiert als Benutzerschnittstelle. Dadurch ist gewährleistet, dass JVx Anwendungen nicht nur am Desktop sondern beispielsweise auch im Web ausgeführt werden können. Durch Schaltflächen auf der grafischen Oberfläche ist eine Interaktion mit dem Server möglich. Durch den generischen Ansatz ist diese technologieunabhängig und bietet die Möglichkeit für die Integration verschiedener Technologien.

Die Kommunikation der einzelnen Schichten erfolgt über eine Remoteverbindung. Der Kommunikationsserver, angesiedelt in der Anwendungsschicht, nimmt die Anfragen entgegen und verarbeitet sie. Als Ergebnis werden die angeforderten Daten an den Aufrufer zurückgeschickt.
Dies ermöglicht es bei wachsenden Anforderungen Skalierungsprobleme zu reduzieren und erhöht gleichzeitig die Wartbarkeit.
Durch den Einsatz von JVx ist es möglich performante und individuell anpassbare ERP Lösungen zu entwickeln.

Unterschied zu ERP Systemen

In Kapitel 3.2 „ERP vs. Verteilte Systeme - Unterschiede in der Architektur“ wurde eine zusätzliche Schicht erwähnt, die Adaptionsschicht.

Ein Framework stellt eine Grundstruktur für die Erstellung von Anwendungen zur Verfügung um die Entwicklung zu vereinfachen (siehe Kapitel 3.3.1).
Adaptionen und sonstige Anpassungen werden von Grund auf berücksichtigt und bei der Umsetzung als ERP System bereits implementiert. Das Produkt ist eine Individualsoftware, zugeschnitten auf einen bestimmten Unternehmensprozess. Daher hat es einen geringeren Anspruch auf Adaptionsmöglichkeiten.

4 JVx.mobile

Auch wenn mobile Devices bereits viele Prozessorkerne besitzen und einiges an Arbeitsspeicher verfügen, können sie nicht mit klassischen Desktop-Computern mithalten - auch in Bezug auf die Datenübertragung. In einem lokalen Netzwerk werden höhere Downloadraten als über mobile Datenverbindungen erzielt.

Um diesen Restriktionen entgegenzuwirken und eine Anwendung zu entwickeln die auf allen neueren Smartphones funktioniert, muss die übertragene Datenmenge optimiert werden und gleichzeitig ein Kompromiss geschlossen werden zwischen geringen Datenmengen, bei einer höheren Anzahl an Requests und einer höheren Datenmenge bei einer geringeren Anzahl an Requests. Der Grund dafür ist die entstehende Latenzzeit pro Request.

Um allen mobilen Geräten einen Zugriff auf JVx-basierte ERP Applikationen zu geben, muss eine plattformunabhängige Kommunikationsschnittstelle zur Verfügung gestellt werden.

Die Lösung dafür war die Entwicklung eines eigenen Servers, der über das Internet Protokoll (IP) Anfragen entgegennehmen kann, die Daten aus der JVx-ERP-Applikation abfragt und diese aufbereitet an den Client zurückschickt.
Dies ist Voraussetzung um in einem weiteren Schritt eine benutzerfreundliche Client-Applikation entwerfen zu können, welche das Hauptziel dieser Master Thesis darstellt.

Im Kapitel „Architektur von ERP Systemen“ wurden Anforderungen, Herausforderungen sowie Designentscheidungen erläutert. Diese werden in diesem Kapitel als Basis für den Entwurf von JVx.mobile herangezogen und anhand der Implementierung belegt.

4.1 Hotspot für die Integration

JVx.mobile ist ein mobiler Server auf Basis von JVx. JVx wurde als JAVA Library importiert und bietet die Schnittstellen für die Kommunikation mit JVx-Applikationen an.
Dieser läuft auf dem Applikationsserver Apache Tomcat.
Die Kommunikation erfolgt über ein RESTful Webservice. Das Kommunikationsprotokoll wird durch ein JSON Dokument repräsentiert.
Durch Verwendung vom HTTP Standard ist eine Integration beliebiger zusätzlicher Mobile Clients möglich. Einzige Voraussetzung ist ein Zugang zum World Wide Web.

Das Ziel dieses Services ist es eine standardisierte sowie technologie- und plattformunabhängige Schnittstelle anzubieten.

Das clientseitig verwendete Betriebssystem sowie die Art des Clients (Smartphone oder Tablet) spielt dabei keine Rolle.
Durch definierte Client-Parameter (siehe Abschnitt 4.4.3) können die Daten optimal für die Darstellung auf mobilen Endgeräten aufbereitet werden.
Der Client fungiert als Terminal und verarbeitet ausschließlich Informationen, die vom Server geschickt werden und konzentriert sich auf eine benutzerfreundliche Darstellung.

Die bestehende Architektur von JVx wurde erweitert und stellt nun eine Schnittstelle zwischen JVx-basierten ERP Anwendungen und den mobilen Endgeräten dar. (Abbildung 4.1)

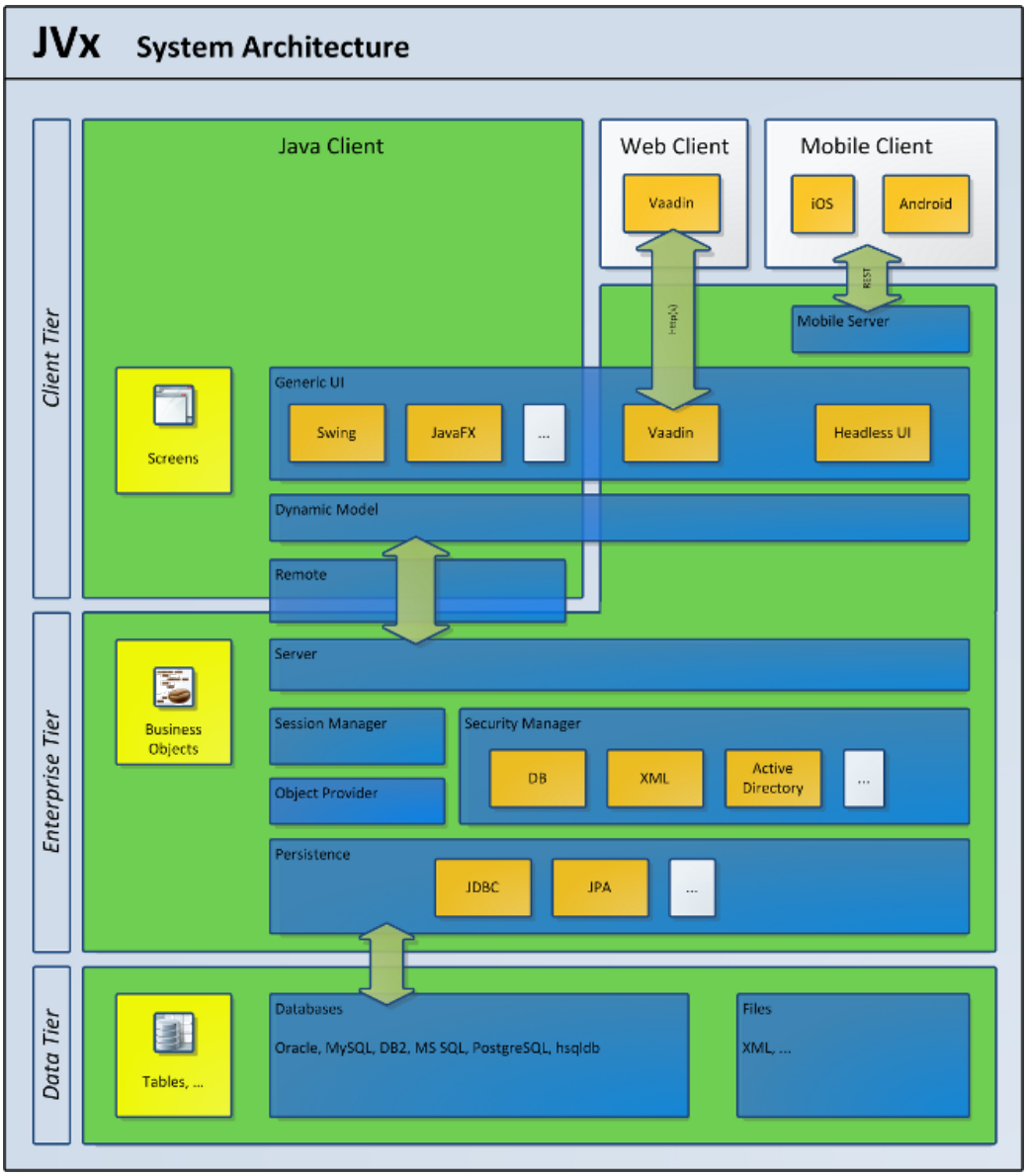

Abbildung 4.1: Architektur nach der Integration von JVx.mobile [28]

Daraus ergeben sich die 2 Hauptziele von JVx.mobile:

- Anbieten einer unabhängigen Kommunikationsschnittstelle
- Geräteabhängige Aufbereitung und Transformation der Daten

Diese Ziele werden im Kapitel 4.4 "Architektur von JVx.mobile“ sowie im Kapitel 4.4.3 „Datenaufbereitung“ diskutiert.

4.2 Designentscheidungen

Ziel ist es dem Benutzer, der Benutzerin eine benutzerfreundliche mobile Clientapplikation zur Verfügung zu stellen. Um das zu erreichen muss bereits bei der Planung und Implementierung des Servers darauf geachtet werden.
Folgende Minimalanforderungen müssen erfüllt sein.

Plattform- und Technologieunabhängigkeit
Das JVx Framework bietet eine Kommunikationsmöglichkeit über ein eigens entwickeltes JAVA-basiertes Kommunikationsprotokoll.
Android Clients können dieses implementieren um mit dem Server zu kommunizieren. Bei iOS Clients sieht es anders aus. Sie basieren auf der Programmiersprache „Objective C“. Für die Verwendung des internen Protokolls müsste dafür ein „Workaround“ mit der „Java Bridge“[12] erstellt werden. Bei Integration anderer Clients müsste dafür wiederum eine eigene Lösung gefunden werden.

Der entwickelte Server soll nicht nur JAVA-basierte Endgeräte (z.B. Android) unterstützen. Vielmehr soll eine standardisierte Lösung geschaffen werden um die Erweiterbarkeit um verschiedener Plattformen zu ermöglichen.

Latenzzeit
Jede Anfrage kostet Zeit. Um dabei auftretenden Latenzzeiten gering zu halten wurde ein Kompromiss geschlossen, die Anzahl an Requests auf ein Minimum zu verringern aber dennoch alle benötigten Informationen an den Client zu schicken.

So unterschiedlich wie jedes mobile Gerät in Größe und Leistung ist, umso vielfältiger sind die Möglichkeiten der Verarbeitung und Darstellung der Daten.
Der Server muss also in der Lage sein auf diese Unterschiede reagieren zu können.

Daraus ergibt sich eine komplexe Herausforderung. Der Lösungsansatz wird im Kapitel 4.4 sowie im Kapitel 4.4.3 vorgestellt.

[12] https://developer.apple.com/

Sicherheit
Sicherheit stellt einen wichtigen Aspekt dar wenn es um die Übertragung von unternehmensinternen und sensiblen Daten geht.
So unterstützt auch JVx.mobile neben dem normalen unverschlüsselten Transferprotokoll (HTTP Verbindung) eine sichere Verbindung über das Internet (HTTPs Verbindung).
HTTPs implementiert dafür eine zusätzliche Schicht, dem Secure Socket Layer (SSL) der für die Verschlüsselung zuständig ist und einen Standard darstellt.

Da HTTP(s) zustandslos ist, müsste sich der Client bei jeder Anfrage neu am Server authentifizieren. Das bedeutet, dass die Zugangsdaten bei jedem Request mitgeschickt werden müssten. Dies stellt ein weiteres Sicherheitsrisiko dar.

Hierfür ist ein erweitertes Sitzungsmanagement notwendig. Es muss gewährleistet werden, dass jede Anfrage einem bestimmten Client zugeordnet werden kann.

Wartbarkeit
In der Master Arbeit wurde die Funktionalität abgegrenzt.
Der Server hat daher die Anforderung drei verschieden komplexe Datenstrukturen zu unterstützen.

Um den Aufwand für zukünftige Adaptionen so gering wie möglich zu halten muss unter Verwendung von Design Patterns eine abstrakte Codestruktur geschaffen werden, die wiederverwendbare Funktionen beinhaltet und beliebig erweitert werden kann.

Der Aspekt der Wartbarkeit muss auch die Tatsache berücksichtigen, dass JVx Applikationen in ihrer Funktionalität wachsen. Der Server muss so aufgebaut sein, dass er ohne großen Aufwand diese neuen Anforderungen unterstützt um sie auf mobilen Devices verfügbar zu machen.

4.3 Plattformunabhängige Kommunikationsschnittstelle

Die Kommunikation stellt eine besondere Herausforderung dar. Sie soll schnell und sicher sein. Das angebotene Service sollte zudem alle mobilen Endgeräte bedienen können, die über eine Internetschnittstelle verfügen.

Um eine einheitliche, standardisierte Kommunikation zu gewährleisten wurde ein Webdienst (engl.: Web Service) implementiert.

Der W3C definiert Web Service folgendermaßen:

„A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." [30]

Ein Web Service ist somit ein „elektronischer Lieferdienst". Er ermöglicht es durch angebotene Schnittstellen auf Daten über das Internet Protokoll zuzugreifen zugreifen zu können. [19]

Somit kann jeder Client der diese Standards versteht auf solche Dienste zugreifen und mit entfernten Systemen kommunizieren. [29]

4.3.1 Aufbau von Web Services

Damit ein passendes Web Service (WS) von einem Client gefunden werden kann und mit ihm kommunizieren kann, muss der Service Anbieter (Provider) seinen Dienst bei einem UDDI[13] Server registrieren. UDDI stellt einen Katalog von verfügbaren WS dar. Er hat die Aufgabe nach Anfrage die URI[14] des gewünschten Web Services zu retournieren um eine direkte Kommunikation zu ermöglichen. [31]

Als nächster Baustein im Web Service Stack wird eine Schnittstellenbeschreibung benötigt. Der Client muss wissen, wie die Anfrage auszusehen hat, welche Funktionalitäten das Service zur Verfügung stellt sowie die Syntax der erwarteten Nachricht und des Resultates. Dies wird durch WSDL[15] beschrieben und stellt somit eine Abstraktion des angebotenen Services dar. Die Beschreibung wird im programmier- und plattformunabhängigem Standard XML definiert. [31]

Um Nachrichten über Netzwerke transportieren zu können wird ein Protokoll benötigt. SOAP[16] hat sich als Standard behauptet. Es bildet den Rahmen der zu übertragenden Informationen und wird ebenfalls im XML definiert. SOAP sind XML-Dokumente gesendet über HTTP, SMTP oder andere Verbindungen. [29]
SOAP bietet keine direkte Kommunikation mit der entsprechenden Funktionalität. Anfragen werden immer an einen Dispatcher gestellt. Dieser entpackt die XML-basierten Nachrichten und liest den Empfänger aus um sie entsprechend weiterzuleiten.

[13] Universal Description Discovery and Integration
[14] Uniform Resource Identifier
[15] Web Service Definition Language
[16] Simple Object Access Protocoll

SOAP ist sehr flexibel. Dadurch können auch Bilder in verschiedenen Formaten übermittelt werden. [31]

WS besitzen keine Benutzeroberfläche. Sie dienen der Vermittlung zwischen Benutzer-, Benutzerinnen Aktionen und einer verteilten Anwendung. [29]

Ein WS kann neben SOAP auch als entfernter Funktionsaufruf (RPC[17]) sowie als „RESTful“ Web Service implementiert werden. [29]

Um das Ziel des mobilen Servers zu erreichen und eine plattform- sowie technologieunabhängige Kommunikation und somit die Einbindung von diversen unterschiedlichen Clients zu unterstützen wurde „REST“ für die Implementierung verwendet. Im folgenden Kapitel wird das Konzept von REST kurz vorgestellt. Die Implementierung in JVx.mobile wird danach in Kapitel 4.4 vorgestellt.

4.3.2 RESTful Web Services

RESTful definiert eine ressourcen-orientierte Architektur.
Die Idee ist abgeleitet von der des Internets. Jede Website, jedes Dokument, jede Information stellt im Internet eine Ressource dar. Diese sind über eine URI identifizierbar.
Über den HTTP Standard kann mit ihr interagiert werden. Mögliche Operationen sind in Tabelle 4-1 ersichtlich.

Definition Ressource

„Eine Ressource ist alles, was wichtig genug ist, um als eigenständiges Etwas referenziert zu werden. Wenn Ihre Benutzer einen Hypertext-Verweis dafür erstellen, Annahmen darüber erstellen oder widerrufen, eine Darstellung davon holen oder speichern, alles oder Teile davon per Referenz in andere Darstellungen übernehmen, es kommentieren oder andere Operationen darauf ausführen wollen, sollten Sie es zu einer Ressource machen.“ [34]
Zusammengefasst stellt jede vorhandene Information eine Ressource dar, mit der interagiert werden kann.

[17] Remote Procedure Call

HTTP Operationen

GET sowie die POST Operation sind essentiell für eine REST-Schnittstelle und stellen die minimalen Anforderungen dar.
Damit ist es möglich Ressourcen abzufragen, neue anzulegen, bestehende zu modifizieren sowie zu löschen.
Die Interaktion mit Ressourcen ist durch folgende Standard HTTP Operationen möglich:

Operation	Verwenden wenn...
HEAD	... nur die Metadaten einer Ressource benötigt werden um Änderungen an dieser zu identifizieren. Gleicher Response wie bei einem GET Request, nur ohne dem Body (Ressource an sich).
GET	... eine bestimmte Ressource angefragt werden soll sowie bei idempotenten[18] Operationen
PUT	... eine Ressource gespeichert werden soll
POST	... wenn eine Ressource angelegt, editiert oder abgefragt (bei einer hohen Anzahl an Parameter) werden muss sowie bei nicht-idempotenten Operationen.
DELETE	... eine Ressource gelöscht werden muss.

Tabelle 4-1: HTTP Operationen [32]

HTTP Codes

Um die Erfüllung oder Nichterfüllung von Requests sowie den Status eines Requests mitzuteilen gibt es standardisierte HTTP Codes. In Tabelle 4-2 werden diese anhand der ersten Stelle des Codes kategorisiert.

Um den Überblick in der Anwendung zu behalten ist es notwendig sich auf ein paar Codes zu beschränken [34].

[18] idempotente Operationen haben keine Seiteneffekte bei wiederholtem Ausführen von Abfragen mit den selben Parametern

Kategorie	Beschreibung
1xx	Diese Kategorie wird selten verwendet. Diese Responsecodes werden lediglich Verwendet um Vereinbarungen zwischen Client und dem HTTP-Server festzulegen. *Vertreter: 100 (Fortsetzen), 101 (Protokollwechsel)*
2xx	Die 2xx-er Reihe repräsentiert erfolgreich durchgeführte Operationen. *Vertreter z.B.: 200 (OK), 201 (Erzeugt), 202 (Angenommen),..*
3xx	Statuscodes dieser Kategorie weisen auf zusätzlich erforderliche Client-Requests um die angeforderte Ressource tatsächlich zu bekommen. *Vertreter z.B.: 300 (Mehrere Auswahlmöglichkeiten), 301 (Permanent verschoben), 302 (Gefunden),..*
4xx	Berichtet auf Clientseite aufgetretene Fehler. *Vertreter z.B.: 400 (Falsche Anfrage), 401 (Nicht autorisiert), 404 (Nicht gefunden),..*
5xx	Berichtet auf Serverseite aufgetretene Fehler. *Vertreter z.B.: 500 (Interner Serverfehler), 503 (Service nicht verfügbar),..*

Tabelle 4-2: Kategorien der HTTP Codes [34]

Schnittstelle definieren

Ressourcen-orientierten Architekturen unterstützen verschiedene Medienformate für die Repräsentation einzelner Ressourcen [33].
Für die Übermittlung als serialisierter Byte-Stream wird meistens das XML Format genommen. Im Web- sowie im AJAX[19]-Bereich wird vorwiegend auf das alternative JSON[20] Format zurückgegriffen. Jenes wird aus assoziativen Arrays aufgebaut. [34]
Für die Entwicklung der Kommunikationseinheit JVx.mobile wurde ebenfalls auf eine JSON-basierte Ressourcenrepräsentation gesetzt.

[19] Asynchrone Datenübertragung (engl.: Asynchronous JavaScript and XML)
[20] JavaScript basierte Notation (engl.: JavaScript Object Notation)

4.4 Architektur von JVx.mobile

Die Entscheidung bei der Wahl der Kommunikationsschnittstelle fiel auf eine RESTful Architektur, da für eine clientseitige Implementierung nur minimale Anforderungen notwendig sind. Jede moderne Programmiersprache ermöglicht das Abschicken von HTTP-Anfragen. Dafür werden unterschiedliche Libraries zur Verfügung gestellt. [32]

Durch den Einsatz dieser Technologie ist es möglich beliebige Clients zu bedienen und stellt durch die minimalen Voraussetzungen eine stabile Lösung dar. Somit kann eine plattform- und technologieunabhängige Kommunikation gewährleistet werden.

4.4.1 Anforderungen an JVx.mobile in Bezug auf die Kommunikation

Gesendete Daten über HTTP Verbindungen werden nicht verschlüsselt. Dies stellt im Internet ein großes Gefahrenpotential dar. Unternehmen werden nur Systeme einsetzen die einen gewissen sicheren Kommunikationsstandard anbieten. Demnach müssen folgende Punkte bei der Implementierung beachtet werden.

Verwenden einer verschlüsselten Verbindung

Daten müssen diskret behandelt werden. Vor allem dann, wenn sie streng vertraulich sind. Die Integrität muss von Anfang an über den ganzen Kommunikationsprozess gewährleistet sein. Dies wird durch den Standard TLS (zusätzliche Sicherheitsschicht bei der Kommunikation, engl.: Transport Layer Security) ermöglicht. HTTP ist ein Schichtenprotokoll. Es basiert auf dem Transportprotokoll „TCP/IP“. Durch Verwendung von TLS ist eine verschlüsselte HTTP Verbindung möglich (HTTPS). Durch den Einsatz von HTTPS-Verbindungen wird Vertrauenswürdigkeit und Datenintegrität gewährleistet. [32]

Zugriffskontrolle

HTTP ist zustandslos. Der Server vergisst sofort nach einem Response wer den Request abgesetzt hat.
Um den Schutz von sensiblen Daten zu gewähren, ist es notwendig, dass eine Zugriffskontrolle implementiert wird. Nur authentifizierte Benutzer, Benutzerinnen dürfen Zugriff auf bestimmte Ressourcen haben. Des weiteren müssen Funktionalitäten für eine Client Autorisierung umgesetzt werden. [32]

Latenzzeiten

Vom Absenden eines Requests durch den Client bis zur Ankunft am Server vergeht eine gewisse Zeitdauer (Latenzzeit). Je nach geografischer Skalierung sowie nach mobiler

Datenanbindung, kann diese länger oder kürzer ausfallen. Weiter ausschlaggebend ist die übermittelte Datenmenge.
Grundsätzlich gilt: je weniger Requests umso geringer ist die Latenzzeit.
Wenn die Datenmenge mitbetrachtet wird, muss dieser Ansatz verworfen werden, da mit erhöhtem Datenaufkommen die Übertragung dementsprechend länger dauert.
Hier ist eine Kompromisslösung notwendig.
Der implementierte Lösungsansatz wird im Kapitel 4.4.2 sowie im Kapitel 4.4.3 behandelt.

4.4.2 Kommunikation mittels Restlet

Für den mobilen Server JVx.mobile wurde auf dem Framework Restlet aufgebaut. Es erleichtert die Umsetzung aufgrund der Vorteile die Frameworks mit sich bringen. (siehe dazu Kapitel 3.3.1)

Restlet stellt Libraries für folgende Technologien zur Verfügung: „Java SE“, „Java EE“, „OSGi“, „GAE“, „Android“ und „GWT“.
Es kann nicht nur für die Implementierung von Servern herangezogen werden sondern auch für die Cliententwicklung. Weiters werden alle HTTP Operationen unterstützt. [33]

Da Restlet unter anderem auf der Android Plattform einsetzbar ist, welche auch auf Java basiert, kann der Code für die Ressource-Repräsentation ohne Änderungen wiederverwendet werden.
Restlet basiert auf einer Open Source Lizenz was die Integration in JVx ermöglicht.
Im Gegensatz zu vielen anderen OS Frameworks steht eine aktive Community dahinter. Die Weiterentwicklung ist somit aus heutiger Sicht gewährleistet.

In Kapitel 3 wurden Anforderungen an verteilte Systeme diskutiert. Die wichtigsten Punkte waren eine gleichzeitige, parallele Benutzung von Ressourcen, eine hohe Skalierbarkeit sowie eine asynchrone, persistente Kommunikation.
Restlet ist thread-safe[21] und für die Nebenläufigkeit ausgerichtet.

Des weiteren werden die in diesem Kapitel erwähnten Sicherheitsanforderungen (Authentifizierung, Autorisierung, Vertraulichkeit) unterstützt.

Der Einsatz von Restlet stellt somit die Umsetzung der Anforderungen sicher und bildet ein stabiles Grundgerüst. Durch die Abstraktion der HTTP Operationen und nichtfunktionalen Anforderungen wird die Komplexität der entwickelten Anwendung gering gehalten.

[21] Threadsicherheit bedeutet, dass Komponenten von mehreren Programmteilen gleichzeitig verwendet werden können ohne dass Werte oder Parameter durch eine andere Instanz überschrieben werden.

Im Folgenden wird der Aufbau von JVx.mobile anhand von Restlet erläutert.

Struktur von Restlet

Der Core des Frameworks stellt die Restlet API dar. Sie ist im Package org.restlet verfügbar und bietet folgende REST Struktur an [33]:

- Uniforme Schnittstelle für die Interaktion mit Ressourcen über Requests und Responses
- Komponenten, die als Container für die Restlet Applikation dienen. Sie beinhalten Server sowie Client Ressourcen
- Konnektoren für die Interkommunikation von REST Komponenten über ein definiertes Protokoll (z.B.: HTTP oder SMTP Konnektor). Durch die Modularität von Restlet können Ressourcen auch auf unterschiedlichen JVM's 22 laufen.
- Repräsentationen von REST Ressourcen

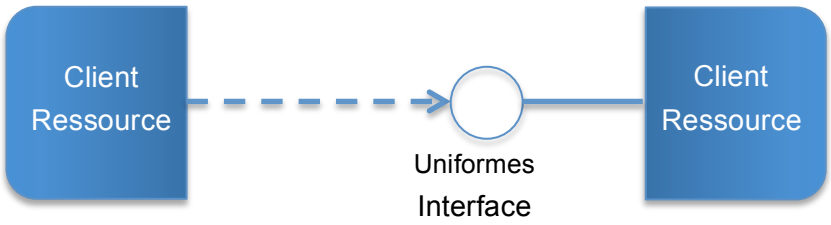

Abbildung 4.2: Repräsentation einer REST Ressource in Restlet [33], eigene Überarbeitung

Eine Restlet Applikation kann sowohl eingehende Serveraufrufe sowie ausgehende Clientaufrufe bearbeiten. Außerdem besteht es aus 3 zentrischen Schichten. Eingehende Anfragen gelangen zuerst an die Service Filter Schicht. Diese ist unter anderem zuständig für die automatische Dekodierung von Repräsentationen. Die eingehende Nachricht wird danach durch eine User-Routing Schicht geleitet. Hier können eigene Filterregeln definiert werden und anhand der Anfrageparameter zu der jeweiligen Ressource weitergeleitet werden. Zudem ist sie für Authentifizierungen verantwortlich. Die letzte Schicht, die durch eine eingehende Nachricht geschickt wird, ist für den Response zuständig. Hier werden Ressourcen aufbereitet und an den Client retourniert. Zusätzlich können Client Requests durch Clientressourcen getriggert werden. [33]

[22] Java Virtual Machine, sie ist zuständig für das Ausführen des JAVA byte Codes und stellt die Laufzeitumgebung einer JAVA Applikation dar.

Aufbau und Architektur von JVx.mobile

JVx.mobile besteht aus einem „`ServerServlet`“ welches als HTTP Konnektor agiert [35] (zu finden im Package: „`com.sibvisions.rad.server.http.rest`“). Es kann mehrere JVx Applikationen gleichzeitig bereitstellen[23]. [35].

Die Konfigurationseinstellungen für das Deployment auf einem Applikationsserver können über eine XML Datei definiert werden. Diese wird standardmäßig als „web.xml“ bezeichnet und befindet sich im folgenden Ordner „/WEB-INF“ am Applikationsserver. Es muss der Pfad entweder zu einer „Applikations-Klasse“ oder einer „Komponenten-Klasse“ eingetragen werden. In JVx.mobile wurde dafür die Klasse `ApplicationAdapter` verwendet. Diese Information wird dafür verwendet um die bereitgestellte Anwendung zu starten. Intern fügt Restlet diese in den Komponenten Kontext ein. In Listing 4-1 ist ein Auszug aus der XML Datei zu sehen.

```
…
<!-- Restlet adapter -->
<servlet>
  <servlet-name>MobileServlet</servlet-name>
  <servlet-class>
       com.sibvisions.rad.server.http.rest.ServerServlet
  </servlet-class>
  <init-param>
    <!-- Application class name -->
    <param-name>
       org.restlet.application
    </param-name>
    <param-value>
       com.sibvisions.rad.server.http.rest.ApplicationAdapter
    </param-value>
  </init-param>
</servlet>
…
```

Listing 4-1: Konfiguration von JVx.mobile für den REST Support

Der Server kennt nun den Startpunkt von JVx.mobile um Requests an JVx.mobile weiterleiten zu können.

[23] Bereitstellen (engl.: Deployment) stellt einen Prozess dar um Anwendungen auf einem Applikationsserver (z.B.: Tomcat) zu installieren

JVx.mobile muss Clientaufgaben wahrnehmen. Er benötigt eine Verbindung zu der JVx Applikation um die benötigten Daten abzufragen, damit er sie aufbereitet an den Absender retournieren kann.
JVx stellt ebenfalls einen Server zur Verfügung der Requests über HTTP(S) entgegennehmen kann. Um diesen Konnex herzustellen muss in der web.xml der in Listing 4-2 gezeigte Abschnitt vorhanden sein.

```
…
<!-- Servlet for accessing the JVx server through http(s)
streaming protocol. -->
<servlet>
  <servlet-name>
       ServletServer
  </servlet-name>
  <servlet-class>
       com.sibvisions.rad.server.http.ServletServer
  </servlet-class>
</servlet>
…
```

Listing 4-2: Konfiguration von JVx.mobile für die Kommunikation mit JVx

Das Servlet ist für die Instanziierung des HTTP Konnektors sowie der JVx.mobile Applikation (`ApplicationAdapter`) zuständig. Hier wird unter anderem der Startpunkt für das Routing festgelegt. In Tabelle 4-3 werden die in JVx.mobile implementierten API Aufrufe beschrieben. Dies stellt die zweite zentrische Schicht des Restlet-Konzeptes dar (siehe Abschnitt „Struktur von Restlet“). In JVx sind folgende Routings definiert, wobei jedes seinen eigenen Handler[24] hat:

API für das User Interface	Beschreibung
/api/StartupCommand	Mit dieser URI wird die vom mobile Client angeforderte JVx Applikation gestartet. Zusätzlich werden Informationen übermittelt die JVx.mobile helfen, die Daten laut Clientanforderungen aufzubereiten. Unter anderem wird hierfür das verwendete Betriebssystem inklusive installierter Versionsnummer, die Bildschirmauflösung, die konfigurierte Spracheinstellung oder der Gerätetyp

[24] Ein Handler stellt einen Prozess dar um ihm zugewiesene Aufgaben zu erledigen

	(Smartphone oder Tablet) übermittelt. Die Transformation der Daten wird im Kapitel 4.4.3 näher behandelt.
/api/LoginCommand	Wird verwendet um sich an der ausgeführten JVx-basierten Anwendung anzumelden.
/api/LogoutCommand	Wird verwendet um sich an der ausgeführten JVx-basierten Anwendung abzumelden.
/api/OpenScreenCommand	Dieser Aufruf wird gemappt und öffnet das angeforderte Programmfenster der JVx Applikation.
/api/GetTableCommand	Der GetTableRequest wandelt die Angeforderte Tabelle vom geöffnetem remote Programmfenster (von der GUI) in eine Tabellenobjekt. Sie können beliebig verschachtelt sein und werden als Master-Detail-Layout aufbereitet an den Client geschickt.
/api/PressButtonCommand	Dieser API Aufruf wird auf eine GUI Aktion gemappt und simuliert einen Maus-Klick auf die remote geöffnete Anwendungsoberfläche.
/api/CloseScreenCommand	Remote geöffnete Anwendungsfenster können darüber wieder geschlossen werden.
/api/ExitCommand	Dieser Aufruf beendet die Applikation und löscht die Clientverbindung.
API für Datenbankaktionen	Beschreibung
/api/BeforeInsertCommand	Diese Schnittstelle ist notwendig, da der Client ein Formular benötigt um zu wissen welche Informationen der Server für den tatsächlichen Insert Befehl benötigt. In der JVx Applikation wird eine „Dummy“ Zeile erstellt um an die notwendigen Informationen zu gelangen.
/api/BeforeUpdateCommand	Beim Editieren von Daten gibt es die Problematik des Primary-Keys. Der Server benötigt den aktuellen Key um den korrekten Datensatz bearbeiten zu können.
/api/InsertCommand	Fügt einen neuen Datensatz in die Tabelle einer Datenbank ein.

/api/UpdateCommand	Ändert einen bestimmten Datensatz in der Tabelle der Datenbank.
/api/DeleteCommand	Löscht einen Datensatz anhand des übermittelten Primary-Keys.
API für Datentransfer	Beschreibung
/download	Anfragen an diese API werden an eine Dokumenten-URI weitergeleitet. Diese schickt den komprimierten, serialisierten Bytestream der Ressource im Response an den Client zurück.
/upload	Eine Schnittstelle für den Dokumentenupload.

Tabelle 4-3: API-Beschreibung von JVx.mobile

Wie in Tabelle 4-3 angeführt, bietet der mobile Server auch Schnittstellen zur Benutzeroberfläche einer JVx-basierten Anwendung. JVx.mobile kommuniziert nicht direkt mit der Enterprise-Schicht von JVx. Stattdessen fungiert der mobile Server als Client und ist in der Client-Schicht angesiedelt.

Verteilte Benutzeroberfläche

Im Kapitel 3.1 wurde erwähnt, dass auch die Benutzerschicht verteilt werden kann.
Der entwickelte Android Client fungiert demnach als Terminal und bietet eine grafische Oberfläche für die Interaktion mit einer beliebigen JVx-basierten Anwendung, welche remote auf dem JVx Server läuft.
Jvx.mobile stellt eine verteilte Benutzeroberfläche (`RemoteWorkScreenApplication`) zur Verfügung.
Sobald ein StartupRequest an JVx.mobile geschickt wird, wird auf der JVM eine Instanz der JVx Anwendung gestartet. Der mobile Client interagiert über JVx.mobile mit diesem Remote-Anwendungsfenster der JVx Applikation.

Um das zu ermöglichen ist es notwendig dieses virtuell geöffnete Anwendungsfenster zu analysieren. Ohne diese Analyse wüsste der Client nicht welche grafischen Elemente er dem Benutzer, der Benutzerin anbieten soll.
Eine Alternative wäre es einen Client für eine spezielle JVx Anwendung zu schreiben. Dadurch wäre eine Applikationsunabhängigkeit nicht gegeben und der Client müsste für jede Anwendung neu entwickelt werden.

Das Kernstück für diese Unabhängigkeit stellt die Analysefunktionalität dar. Sie befindet sich im Package „`com.sibvisions.rad.server.http.rest.analyze`". Die Hauptaufgabe ist die Analyse der am JVx Server geöffneten Work-Screens. Der Output wird im Response zusammengefasst und an den Client geschickt.

Er weiß nun welcher Screen gerade angezeigt werden muss und welche Tabellen in welcher Verschachtelungstiefe vorhanden sind. Das Set an ausführbaren Aktionen kann ebenfalls abgefragt werden.
Dadurch ist es möglich einen Anwendungsunabhängigen Client zu entwickeln.

Eine weitere wichtige Aufgabe der Analysefunktionalität stellt die Transformation der Daten anhand der Client-Parameter dar. Die Datenaufbereitung stellt neben einer offenen Kommunikationsschnittstelle das zweite Ziel von JVx.mobile dar und wird im Abschnitt „Datenaufbereitung" behandelt. Sie ist Voraussetzung um am mobilen Client eine benutzerfreundliche grafische Oberfläche bereitzustellen. Diese Darstellung ist das Hauptziel dieser Master Thesis.

Für die Kommunikation und Repräsentation der Ressourcen musste ein Standard definiert werden. Da es um Datenbankapplikationen geht wurde die objektorientierte Datenrepräsentation anhand der verschachtelten Datenbanktabellen gewählt.

SimpleTableScreen

Der einfachste Fall stellt ein Anwendungsfenster mit einer einfachen Tabelle und verschiedenen Interaktionsmöglichkeiten dar. Das heißt es gibt keine verschachtelten Datenstrukturen.

ComplexTableScreen

Hierbei handelt es sich um einen komplexeren Screen. Dieser wird verwendet wenn sich auf der Remote-Anwendung eine GUI mit mehr als einer Tabelle befindet. Auch Master-Detail Tabellen werden unterstützt. Die Detailtabellen eines Datensatzes müssen allerdings in einem gesonderten Request angefordert werden um eine schnelle und stabile Kommunikation zu ermöglichen.

Beide Ausprägungen enthalten neben ein oder mehreren Tabellen auch GUI Elemente.
Jedes Objekt ist ein serialisierbares „`AbstractBean`". Dies erleichtert die Transformation in eine JSON-Notation die als Response beim Client eintrifft und einfach deserialisiert und in ein entsprechendes Objekt umgewandelt werden kann. Das für den Transport definierte Protokoll ist JSON und wird im folgendem Abschnitt behandelt.

Diese zwei Repräsentationen reichen aus um einen Großteil der möglichen Datendarstellungen in JVx zu mappen. Somit kann der Client ohne Änderungen für Anwendung A oder Anwendung B gleichermaßen verwendet werden. Das ist ein Vorteil der Anwendungsunabhängigkeit.

Im Hinblick auf die im Kapitel 3.1 erwähnten Designentscheidungen verteilter Systeme wurden folgende Konzepte implementiert um diesen gerecht zu werden.

Die Wartbarkeit sowie Erweiterbarkeit sind eng miteinander verbunden und beeinflussen sich gegenseitig. Alle Objekte basieren auf einem `Bean`-Konzept. Beans sind Software-Konstrukte die einfach instanziiert werden können und durch deren Serialisierbarkeit leicht für den Datentransport aufbereitet werden können. Sie dienen als Grundlage für alle Requests und Responses.

In JVx werden sie `AbstractBean`s genannt. Deren interne Struktur basiert auf einer JAVA Map[25]. Sie ist sehr flexibel und kann beliebige Objekte speichern. Zu finden ist sie im Package „`com.sibvisions.rad.server.http.rest.bean`". Dieser Aufbau bietet somit eine leichte Wartbarkeit. Durch die generisch gehaltene abstrakte Klasse ist diese beliebig erweiterbar. Auf Grund des Konzeptes der Vererbung, können beliebige Datenrepräsentationen abgeleitet werden ohne zusätzliche Änderungen und Adaptionen am Core von JVx.mobile.

Um den Client-Request einer bestimmten ausgeführten JVx Anwendung zuordnen zu können, muss bei jeder Anfrage, die am Anfang vom Server berechnete Client-ID mitgeschickt werden. Dies ist ebenfalls in der abstrakten Repräsentation enthalten. Das ist notwendig für die Implementierung der Autorisierung.

JSON als Protokoll

Der Austausch von Ressourcen erfolgt über das JSON Format. Durch das Bean-Konzept ist das Mapping in eine JSON Repräsentation einfach möglich. Dieses Mapping wurde auch am Android Client übernommen. Der Aufbau konnte so wiederverwendet werden.
JSON ist eine objektbasierte JavaScript Notation und vergleichbar mit assoziativen Arrays. Diese können beliebig tief verschachtelt werden.

Jedes GUI Element ist eine Komponente und hat eine eindeutig zugewiesene ID. Das eindeutige Identifikationsmerkmal ist Grundlage für die Analyse der geöffneten Screens und somit der Datengewinnung.

[25] eine Map ist ein Objekt welches anhand von Schlüssel-Werte-Paaren beliebig Daten speichern kann.

Da der Client als Terminal fungiert und einen generischen Aufbau besitzt, benötigt er eine Information welche Vorlage (Template) er laden muss, damit die ihm geschickten Informationen dargestellt werden können.
Dies kann beispielsweise „`screen.simpleTable`“ oder „`screen.complexTable`“ sein. Diese Vorlagen definieren das grundlegende Layout der clientbasierten Darstellung.

Das JSON teilt sich weiter in einen Teil „`table`“. Er enthält Spalteninformationen (z.B. den Spaltennamen laut Datenbank, Rendering Informationen („`cellEditor`“) oder ob der Wert editierbar sein soll). Die tatsächlichen Werte werden im JSON-Tag „`rows`“ eingefügt.

Der Aufbau des `ComplexTableScreen`s ist ähnlich. Statt einer Tabelle („`table`“) werden hier mehrere Tabellen („`tables`“) mitgeschickt. Zusätzlich können mehrere Detailtabellen („`detailTables`“) vorhanden und beliebig tief verschachtelt sein.

Zusätzliche Parameter wie „`visibleColumnCount`“ oder „`visibleColumns`“ werden für eine optimierte Darstellung verwendet und anhand der Clientparameter auf ein sinnvoll darstellbares Set reduziert. (mehr dazu im Abschnitt Clientparameter)
Listing 4-3 zeigt die JSON Struktur anhand des `SimpleTableScreen`s.

```
({
componentId = P7601;
name = "screen.simpleTable";
  table = {
    cells = (
              {
              cellEditor = {
                 name = NumberCellEditor;
                 };
              columnName = KUNDEN_NR;
              dataTypeIdentifier = 3;
              editable = 1;
              value = "<null>";
              }
    ),
    componentId = T7602;
    insertEnabled = 1;
    updateEnabled = 1;
    deleteEnabled = 1;
    detailTables = ();
    labels = (Kundennummer);
    primaryKeyColumns = (KUNDEN_NR);
    rows = ((14), (15), (16));
    visibleColumnCount = "-1";
    visibleColumns = ("KUNDEN_NR ");
  };
})
```

Listing 4-3: JSON-Repräsentation vom SimpleTableScreen mit einer Tabellenspalte

Ablauf Request/Response

Anhand eines einfachen Szenarios wurde der Kommunikationsablauf durch ein Sequenzdiagramm visualisiert. In Abbildung 4.3 wird der Startprozess rudimentär beschrieben. Er zeigt auch, dass JVx.mobile direkt mit dem UI der gestarteten Applikation kommuniziert und sich auf seine Hauptaufgaben konzentriert: Abfrage, Analyse sowie Aufbereitung der erhaltenen Daten von der remote gestarteten Applikation.

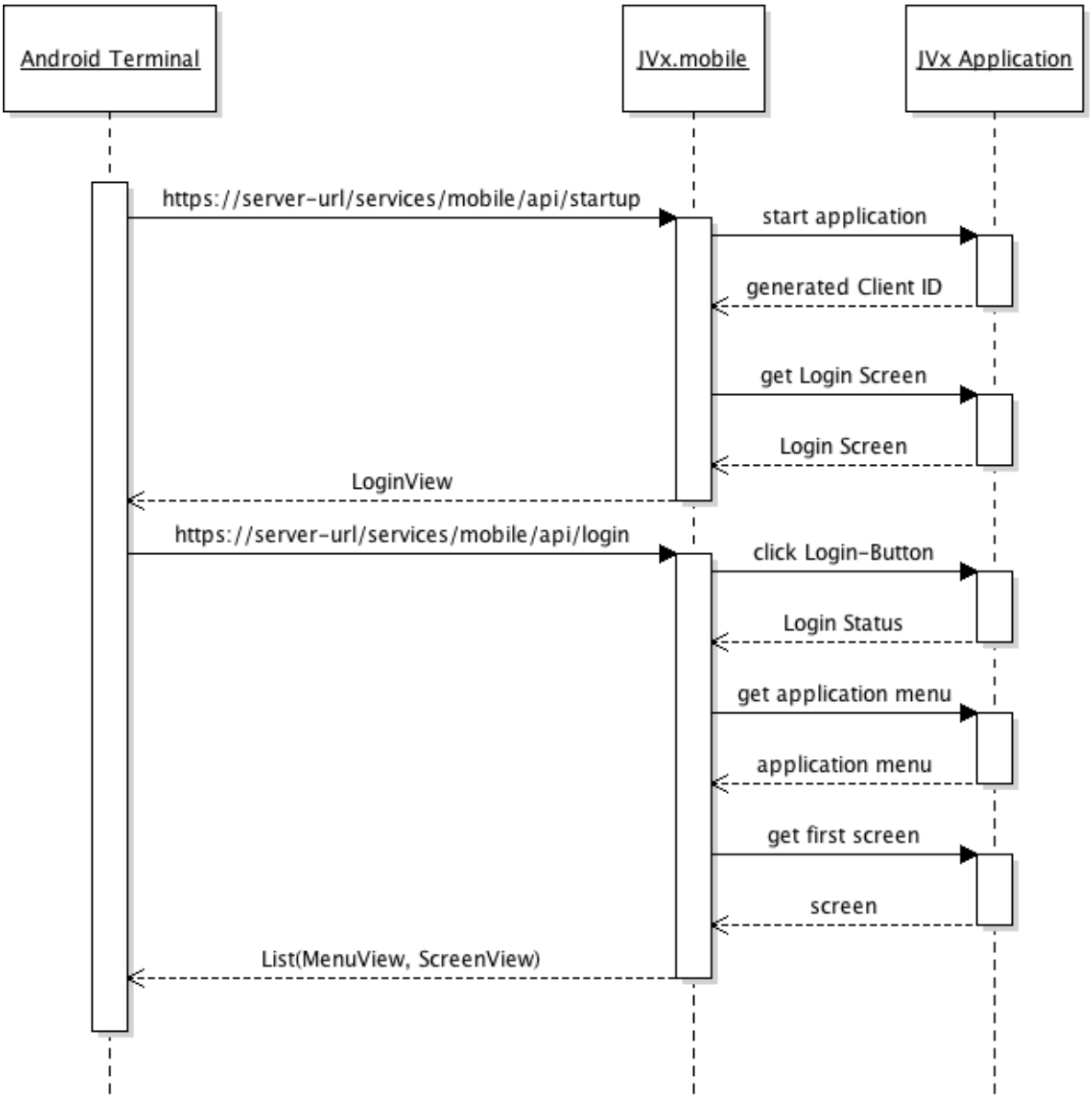

Abbildung 4.3: Starten einer Applikation und Interaktion mit der Remoteanwendung

Auch die Sessionverwaltung sowie die Authentifizierung erfolgt direkt über die JVx-Applikation. Anhand der generierten Client-ID und dem Authentifizierungsschlüssel die bei jedem Request mitgeschickt werden, ist eine eindeutige Zuordnung zu der gestarteten Anwendung möglich. Der Client muss seine Login Daten nicht jedes Mal mitschicken.
Der Server merkt sich zudem auch mitgeschickte Client-Parameter. Aufgrund dieser basiert die Analyse und Datenaufbereitung.

Die Server-Antwort kann auch aus einer Menge von „Screens“ bestehen. Dies ist notwendig um die Zahl der Requests zu minimieren.
Der Login-Response enthält eine `MenuView`. Diese enthält alle Menüpunkte die für eine Interaktion am Client freigeschalten wurden und sind mit denen der Remoteanwendung

ident. Menüeinträge werden auf Android in einem Kontext-Menü dargestellt (siehe Kapitel 5). Dies hat zur Folge, dass der restliche Bildschirm leer ist. Hier wäre zwingend ein neuer `GetTableRequest` notwendig, welcher Daten für die Anzeige liefert.

Um dem entgegenzuwirken wird zusätzlich nach erfolgreichem Login eine definierte „Start-Tabellenansicht“ mitgeschickt. Es entfallen zusätzliche Latenzzeiten.
Der `MenuView` Response hat aufgrund seiner geringen Daten ebenfalls kaum Einfluss auf die Größe der übermittelten Daten.

Im Kapitel 5.3 „Usability von MyERP“ werden unter anderem die daraus resultierenden Vorteile aus der Usability-Sicht betrachtet.

4.4.3 Datenaufbereitung

Eine wichtige Eigenschaft von JVx.mobile ist es Daten so aufzubereiten dass sie auf verschiedenen Endgeräten optimal dargestellt werden können.
Um dies zu erreichen werden bei der Startup Anfrage geräteabhängige Merkmale mitgeschickt (siehe Abschnitt Clientparameter).
Der Server kann anhand dieser die Daten optimal aufbereiten.

Clientparameter

Um eine auf den Client zugeschnittene Datenaufbereitung zu garantieren muss der Client sich mit folgenden Informationen am Server registrieren. Diese Parameter werden im `StartupRequest` mitgeschickt. In folgender Auflistung werden diese mit den entsprechenden JSON-Parametern erläutert.

- **Applikationsname -** `applicationName`
  Der Client gibt hier den Applikationsnamen an auf die er zugreifen will.
- **Version der Clientanwendung -** `appVersion`
  Um Änderungen in den Sprachdateien oder Grafiken am Client erkennen zu können muss die definierte Versionsnummer über die Schnittstelle mitgeliefert werden.
- **Betriebssystem -** `osName`
  Hier können Daten anhand des verwendeten Betriebssystems entsprechend angepasst werden.
- **Version des Betriebssystems -** `osVersion`
  Neben dem Betriebssystem kann JVx.mobile auch die Version dessen speichern und für die Analyse verwenden.
- **Bildschirmauflösung –** `screenHeight, screenWidth`

Je nach Bildschirmgröße des Clients kann der Server entscheiden wie viele Informationen an den Client gesendet werden um die übermittelte Datenmenge zu optimieren und gering zu halten.

- **Gerätetyp –** `deviceType, deviceTypeModel`
  Hier kann zwischen einem Tablet und einem Smartphone unterschieden werden um den Response auf dessen Spezifika anzupassen. Sowie dem Type (bspw.: iPhone oder iPod)
- **Spracheinstellung des Clients -** `langCode`
  Je nach Spracheinstellung werden dem Client automatisch die Daten entsprechend der Sprache aufbereitet und er erhält das personalisierte Sprachpaket.

Diese bilden die Grundlage für eine optimale Analyse und Aufbereitung. Er verwaltet die `ApplicationResource`, den `ApplicationRequest` sowie den `MobileLauncher` und implementiert die Logging Funktionalität (`ILogger`). Die Klasse definiert des weiteren das Interface für die konkretisierten Klassen (siehe Abbildung 4.4). Die Abstrakte Klasse gibt vor welche Informationen für die Instanziierung benötigt werden und stellt die Methodendeklaration für die GUI-Analyse (siehe Listing 4-4) zur Verfügung.

```
public abstract void analyze(List<ApplicationResponse>
pResponse)
```

Listing 4-4: Interface für die GUI Analyse und der Response-Aufbereitung

Von den Anforderungen für JVX Applikationen wurden die in Abbildung 4.4 benötigten Klassen abgeleitet.

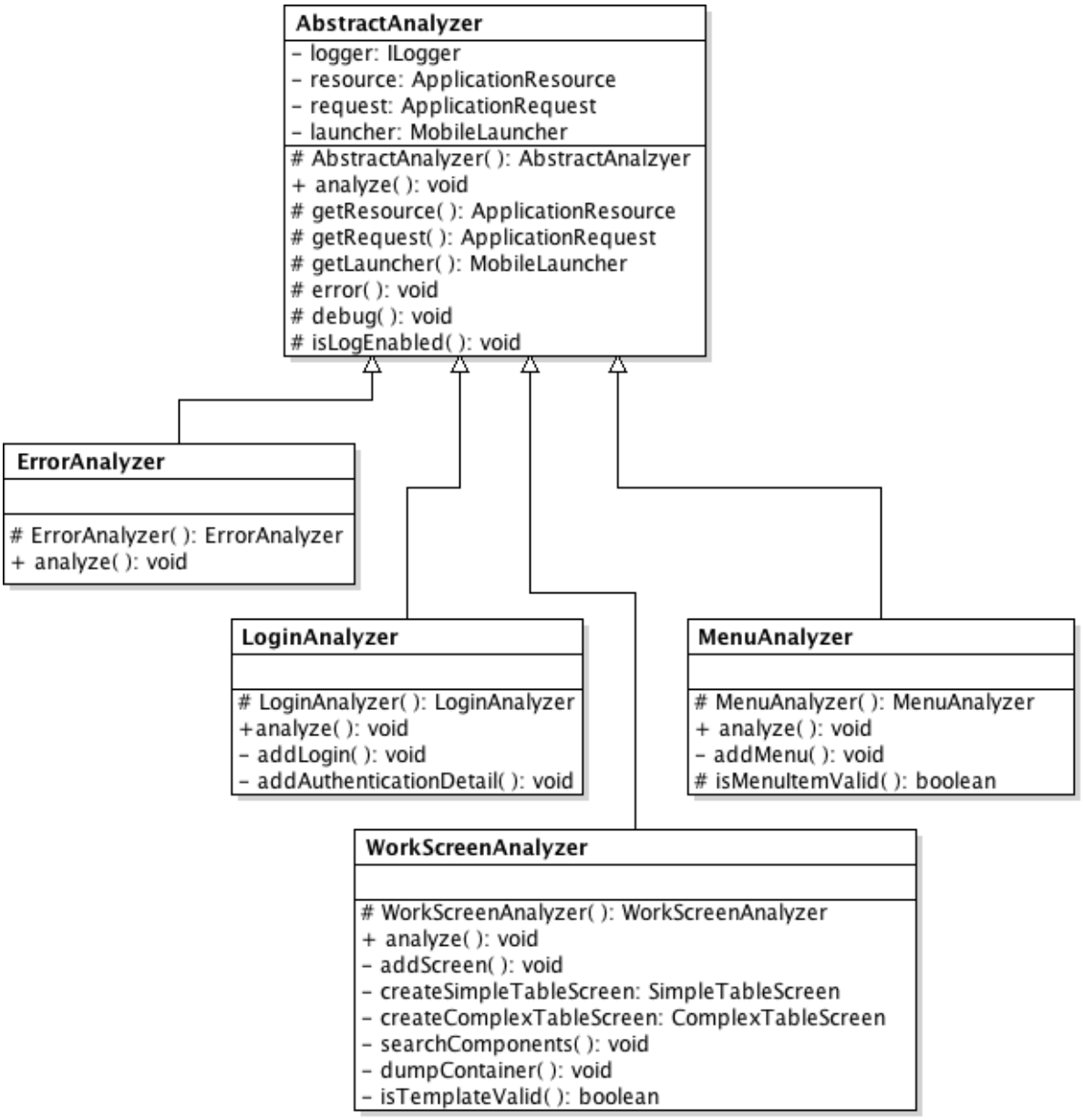

Abbildung 4.4: Klassendiagramm der Analyseklassen

Der `WorkScreenAnalyzer` ist für die Analyse aller datenbehafteten Work-Screens zuständig und fügt jeden einzelnen zum Response hinzu.

Ablauf der GUI Analyse und dessen Datenaufbereitung

Zuerst wird jeder Screen einzeln eingelesen, nach allen sichtbaren UI-Elementen gesucht und in einer `ComponentGroup` zusammengefasst (siehe [38]).
Dies bietet die Grundlage für die eigentliche Analyse und der Aufbereitung des Responses.

Grundsätzlich entscheidet die Methode `addScreen` ob es sich um eine einfache Tabelle (`SimpleTableScreen`) oder um eine komplexere Ansicht mit mehreren Tabellen (`ComplexTableScreen`) handelt.

Anhand dieser Entscheidung werden die gefundenen Datentabellen, auf die Clientbedürfnisse angepasst, aufbereitet. Diese Funktionalität wurde in eine Hilfsklasse `Util` ausgelagert, welche im Package `com.sibvisions.rad.server.http.rest` zu finden ist. Die dafür zuständige Methode ist `createTable`.
Sie holt sich die zu transformierende Tabelle aus der JVx Applikation über das Interface `IDataBook` (siehe [37] für weiterführende Informationen) und bereitet sie für den mobile Client auf. Neben der Tabelle werden auch alle für mobile Endgeräte relevanten Tabellenaktionen abgefragt (Einfügen, Bearbeiten, Löschen).

Der Server bereitet die Daten anhand der Clientsprache auf und formatiert diese dementsprechend. JVx liefert die einzelnen Daten im JVx Format. Da der Client als Terminal fungieren soll, ist auch eine entsprechende Aufbereitung und Transformation in eine Zeichenkette (String) notwendig. Diese kann der Client ohne Typenumwandlungen und –konvertierungen dem Endbenutzer, der Endbenutzerin ausgeben.

Für die Berechnung und Entscheidung, welche Daten auf dem jeweiligen spezifischen Endgerät sinnvoll dargestellt werden können, werden folgende 2 Interfaces bereitgestellt. Diese können durch den Applikationsentwickler herangezogen werden um anhand der Auflösung und anderen Bildschirmeigenschaften das benötigte Subset zu berechnen.

`IMobileTableControl` ist ein Interface, abgeleitet von `ITableControl` (JVx). Dieses stellt Methoden zur Verfügung um das aktuelle `DataBook` [26]abzurufen und neu zu setzen. In JVx.Mobile kann es verwendet werden um sichtbare Tabellenspalten für mobile Geräte zu berechnen.

Eine Datentabelle kann aus mehreren Einträgen (Zeilen) bestehen. Jede Zeile kann mehrere Spalten haben. Auf Desktopanwendungen oder im Web können sie einfach ausgegeben werden. Auf mobilen Devices herrscht Platzmangel. Daher muss eine entsprechend praktikable Darstellung gewählt werden. Auf diesen Sachverhalt wird im späteren Kapitel „MyERP - Mobiler Android Client“ eingegangen.

Um den Client dahingehend programmieren zu können bietet JVx.mobile das Interface `IMobileTableOverview` an. Hiermit können Spalten für die Vorschau am Client definiert werden.

[26] IDataBook stellt eine speicherunabhängige Datentabelle dar. Sie verfügt über die CRUD-Datenbankaktionen und beinhaltet die einzelnen Tabelleneinträge. [37]

Bevor diese ermittelten Daten an den Client retourniert werden, wird der Layout Parameter gesetzt.
Da der Client keine Informationen über die Applikation an sich hat, benötigt er diese um zu entscheiden welches Template er für die Darstellung verwenden muss. Welche Auswirkungen und Möglichkeiten das Template bietet, wird im folgendem Kapitel näher erläutert.

Optimierungspotenzial

Während der Implementierung wurde folgendes Szenario diskutiert, wozu auch ein Lösungsvorschlag ausgearbeitet wurde.

Ab einer gewissen Datenmenge kommt es clientseitig zu einem Leistungsabfall bis hin zu Speicherplatzproblemen. Je nach verbauter Hardware können unterschiedlich viele Datenzeilen einer Tabelle am mobilen Gerät verarbeitet werden. Bis zu dieser Grenze ist eine flüssige Benutzerinteraktion gewährleistet.
Serverseitig ist es möglich die übermittelten Daten zahlenmäßig zu begrenzen. Eine zukünftige Optimierung wäre die Implementierung von „Lazy Loading“ und/oder einer Art Paginierung.

Lazy Loading stellt ein Design Paradigma dar. Fowler beschreibt ihn folgendermaßen:

„An object that doesn't contain all of the data you need but knows how to get it.“ [39]

Das bedeutet, dass nicht alle Daten auf einmal in den Speicher geladen werden sondern nur dann, wenn sie explizit angefordert werden. Um dies zu bewerkstelligen wird ein Marker an der letzten Position gesetzt. Somit ist die Information des letzten geladenen Datensatzes vorhanden. Es gibt verschiedene Möglichkeiten dieses Pattern umzusetzen. [39]

Um den Client noch generischer gestalten zu können wäre es möglich, das auf XML basierte Layout am Server generieren zu lassen und so ein Mapping zu schaffen um grafische Änderungen automatisch auf alle mobilen Geräte zu übertragen.

Die Unterstützung von Uploads sowie dem Rendering von Diagrammen wären weitere Maßnahmen um den Anforderungen heutiger ERP Systeme gerecht zu werden.

Die Konzepte sind nur als Idee für JVx.mobile vorhanden. Über eine genauere Implementierung wurde noch nicht nachgedacht und würde den Rahmen der Master Thesis übersteigen.

5 MyERP - Mobiler Android Client

Der mobile Client wurde anwendungsunabhängig entwickelt. Um die Funktionalität testen zu können, hat SIB Visions eine ERP Anwendung („MyERP“) entwickelt mit den gängigen Funktionalitäten. Wie im vorigen Kapitel erwähnt fungiert der Client als Terminal.
Um eine benutzerfreundliche Bedienung sicherzustellen wurde als oberste Schicht eine grafische Oberfläche entwickelt.
Der Server steuert und gibt vor welche Elemente dem Benutzer, der Benutzerin angezeigt werden. Der Client wurde daher generisch entwickelt um höchstmögliche Flexibilität zu gewährleisten.

In diesem Kapitel wird das Endergebnis dargestellt und auf das Thema Usability eingegangen sowie Designentscheidungen reflektiert.

5.1 Herausforderungen

Bei der Entwicklung des Client gab es einige Herausforderungen zu bewältigen.

Das erste Thema war die Version der **Runtime Target-API** abzustecken. Das Ziel war die meist verbreiteten Android Versionen herauszufinden um eine gewisse Kompatibilität zwischen den Versionen zu gewährleisten. Der aktuelle Stand der Verteilung ist in Abbildung 5.1 zu sehen.

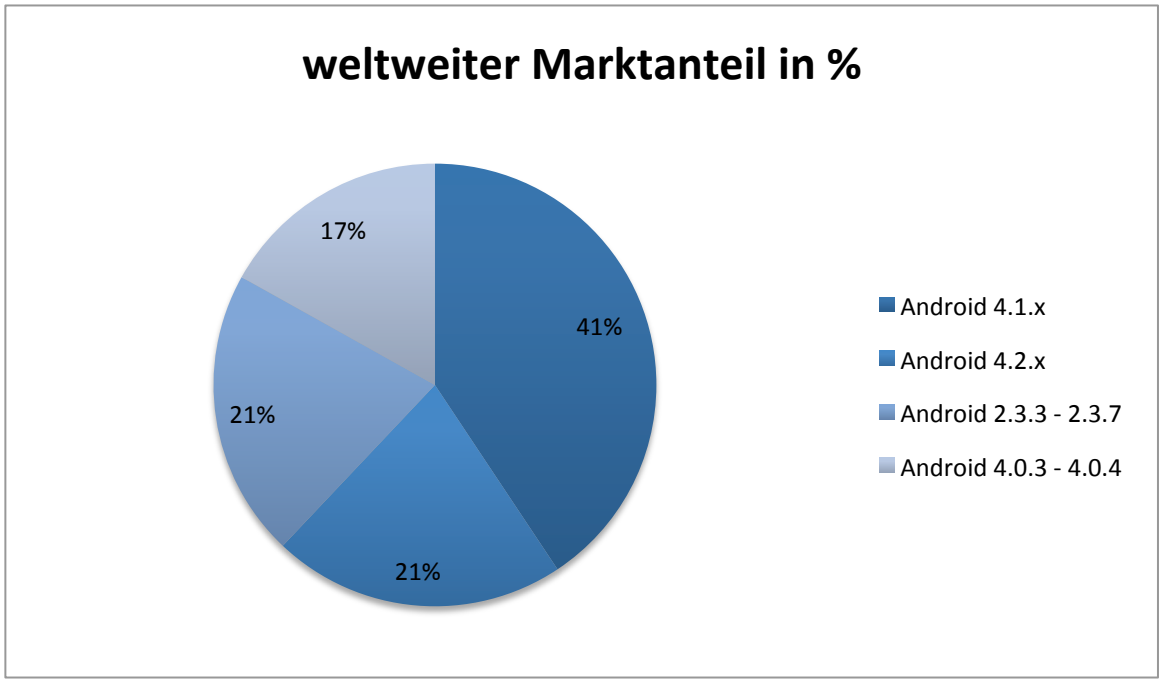

Abbildung 5.1: Verteilung der Android-Versionen am weltweiten Markt[36], eigene Überarbeitung

Da die Version Android 2.3.3 noch recht verbreitet ist wurde dessen API Version 10 als die unterste Grenze an unterstützten Android-Geräten gewählt. Damit wird ein Großteil der Android Devices unterstützt.

Daraus abgeleitet entsteht die nächste Herausforderung. Da es zwischen der Version 2.3.3 und 4.x einige grundlegende Änderungen gab, musste bei der Entwicklung des Android Clients auf diese eingegangen und etwaige Workarounds gefunden werden um diese **Abwärtskompatibilität** zu gewährleisten.

Da es diverse Gerätehersteller gibt und diese **unterschiedlichste Hardware** verbauen, stellt die Bildschirmgröße sowie die Pixeldichte (dpi) ein weiteres Problem dar. Auch die Prozessorleistung muss bei der Verarbeitung großer Datenmengen berücksichtigt werden.

Die zwei Hauptherausforderungen sind erstens das Design sowie dessen **anwendungsunabhängige Implementierung** und zweitens eine **benutzerfreundliche Darstellung** am Bildschirm.

In diesem Kapitel wird die Anwendungsunabhängigkeit behandelt sowie Bezug auf die Usability genommen.

5.1.1 Designentscheidungen

Die Designentscheidungen wurden von den Herausforderungen sowie den Anforderungen abgeleitet.

Für eine **Applikationsunabhängigkeit** muss der Client generisch aufgebaut werden.
Die clientseitige Architektur dafür ist das Eine, aber ohne Unterstützung von JVx.mobile ist das nicht realisierbar.

Das Aufbereiten des UI darf nicht der Client übernehmen. Er soll maximal verschiedene Layouts zur Verfügung stellen, aus denen der Server wählen kann, welches für die Darstellung verwendet werden soll.

Der Server muss entscheiden welche UI Elemente angezeigt werden, in welcher Reihenfolge sowie welche UI-Aktionen nach Benutzerinteraktion ausgelöst werden sollen. Diese Entscheidung trifft der Server durch die integrierte Analysefunktion unter Berücksichtigung der Client-Eigenschaften. Der Aufbau dieser wurde im Kapitel 4.4.3 behandelt.

Die für einen applikationsunabhängigen Client notwendige Architektur wird im nächsten Unterkapitel dargestellt.

Um das Android Terminal **benutzerfreundlich** gestalten zu können, wurden bestehende Standard-Android Anwendungen untersucht (Kalender-, Mail-, Kontakte-Programm). Basierend auf diesen wurden Interaktionsabläufe festgelegt. Durch die zusätzliche Verwendung von Android Vorgaben und der Styleguides wurden auf den Android Pattern optimierte Anwendungsoberflächen entwickelt.

Aufgrund der noch großen Verbreitung älterer Android Versionen stellt die **Abwärtskompatibilität** eine große Herausforderung für jeden Android-Entwickler, jede Android-Entwicklerin dar. Mit jeder neuen Version wurden Teile der Android Runtime optimiert, erweitert oder durch neue Komponenten ersetzt.
Auch Handy-Hersteller sind davon betroffen.
Jeder Hersteller möchte seine eigene Corporate Identity wahren indem er eine eigene „Look And Feel“ Anwenderoberfläche in den Android-Stack integrieren.
Daher können sie auf der einen Seite den Wartungsaufwand neuer Android-Versionen nicht tragen um jedes am Markt befindliche Gerät upzudaten bzw. zu supporten. Auf der anderen Seite stellen diese herstellerabhängigen User-Interface-Implementierungen weitere Herausforderungen dar.
Auch der Endbenutzer, die Endbenutzerin tut sich schwer bei diesen verschiedensten Versionsnummern den Überblick zu behalten.

Eine Lösung für Entwickler und Entwicklerinnen stellt Google selbst zur Verfügung. Durch die „`Android Support Library`“ wird eine Abwärtskompatibilität gewährleistet. Sie stellt ein „Funktionsupgrade“ dar. Nicht vorhandene Funktionen in älteren Versionen werden dadurch nachgerüstet. Jede Android API Version benötigt eine spezielle Version dieser Library.
Für diese Arbeit wurde die Revision 12 verwendet. (siehe Abschnitt 0 für nähere Informationen)

Auf **Unterschiede in der Hardware** (z.B.: die Display-Größe) muss ebenfalls eingegangen werden um einen maximalen Output zu erlangen. Diese Unterstützung übernimmt JVx.mobile und wurde im Kapitel 4 Abschnitt „Datenaufbereitung“ bereits behandelt.

Leistungsunterschiede einzelner Android Devices werden in dieser Arbeit nicht berücksichtigt. Eine mögliche Optimierung wurde im Kapitel 4.4.3 behandelt.

Aufgrund dieser Designentscheidungen ergibt sich die im nächsten Abschnitt „Architektur des Android Client“ beschriebene Architektur.

5.2 Architektur des Android Client

Android Anwendungen bestehen aus Komponenten. Diese sind lose gekoppelt und durch das Applikations-Manifest (`AndroidManifest.xml`) verbunden. Es beschreibt wie die einzelnen Komponenten miteinander interagierten. Zusätzlich können applikationsabhängige Metadaten definiert werden, wie zum Beispiel welche Android API Version unterstützt werden soll sowie benötigte Berechtigungen und die Registrierung der „Activities“. [41]

Android besteht aus folgenden Komponenten [41]:

- **Activities**
  Diese Komponenten bilden die Präsentationsschicht, das User Interface. Es besteht aus Fragmenten und Views um das Layout zu bestimmen.

- **Services**
  Services arbeiten im Hintergrund und besitzen keine eigene grafische Oberfläche. Sie sind zuständig für Aufgaben die asynchron abgearbeitet werden müssen und dementsprechend mehr Zeit in Anspruch nehmen. Der User Interface Thread wird dabei nicht blockiert.

- **Content Providers**
  Sie bilden eine Persistierungsschicht. Darin können anwendungsspezifische Daten gespeichert oder beispielsweise eine Anbindung an SQL Datenbanken definiert werden. Da jede Anwendungsinstanz in einer eigenen VM läuft, kann das Konzept des Content Providers dazu genutzt werden um mit anderen Anwendungen zu kommunizieren.

- **Intents**
  Intents bilden ein „Nachrichten-Transfer Framework“ für die Interkommunikation innerhalb von Android Anwendungen.

- **Broadcast Receivers**
  Diese Komponente stellt einen Listener dar, die auf Intents reagieren kann und dementsprechende Events triggert.

- **Widgets**
  Ein Widget stellt eine dynamische, interaktive, visuelle Anwendungskomponente dar, die auf dem Home-Screen des Android Devices hinzugefügt werden kann. Sie basiert auf einem Broadcast Receiver.

- **Benachrichtigungen**
  Durch dieses Konzept kann der Benutzer, die Benutzerin über Änderungen oder sonstige Aktivitäten informiert werden. Dies funktioniert sowohl wenn sich die Anwendung im Hintergrund befindet oder auch komplett geschlossen wurde.

5.2.1 JVxApplication – der Einstiegspunkt

Im Folgenden bezieht sich das Wort „Client“ auf die entwickelte JVx-basierte Android Anwendung.

Den Einstiegspunkt jeder Android Anwendung bildet die `Application` Klasse. Jede Anwendung hat genau eine Instanz davon (Singleton).

Für die in dieser Master Thesis verwendete Demo-Anwendung „MyERP“ wurde die abgeleitete Klasse `JVxApplication` erstellt. Sie dient als zentraler Speicher für Anwendungsdaten sowie –einstellungen, die von mehreren Komponenten benötigt werden. Sie wird des weiteren für den Transfer von Objekten zwischen einzelnen Komponenten verwendet.

Auch der Applikations-Lifecycle kann überschrieben werden um beispielsweise Initialisierungen diverser Parameter beim Applikationsstart vorzunehmen.
`JVxApplication` stellt somit einen Container dar, der über den aktuellen Status der Anwendung Bescheid weiß. Dieser Einstiegspunkt muss ebenfalls in der Manifest Datei deklariert werden [41].

5.2.2 Grafische Oberfläche

Wie vorhin erwähnt stellt die `Activity` Komponente die Basis für die grafische Anwendungsoberfläche dar.

Seit Android 3.0 ist die bevorzugte Weise Screens zu gestalten mittels Fragments, Layouts und Views. Um auch ältere Android API Versionen zu unterstützen wurde die im vorigen Kapitel angesprochene „Android Support Library“ verwendet. Da die neue Actionbar nicht unterstützt wird wurde zusätzlich das Projekt ActionBarSherlock[27] importiert für die Abwärtskompatibilität des Menüs.

Activities

Für eine Applikationsunabhängigkeit enthält der Client folgende generische Activities.

[27] http://actionbarsherlock.com

LoginActivity bzw. MainMenuActivity sind konkrete Klassen, die der Client unterstützen muss. Das sind fixe Bestandteile einer jeden JVx Anwendung.

Jede dieser Activities basiert auf einer Basisklasse `BaseFragmentActivity`. In der Lifecycle Methode `onCreate` wird die Initialisierungsreihenfolge jeder Activity festgelegt. Des weiteren bietet sie die dafür notwendigen Interfaces als abstrakte Methodendeklarationen an (siehe Listing 5-1). Dadurch erhöht sich die Wartbarkeit des Clients und ermöglicht es neue Activities nach definierten Standardregeln hinzuzufügen.

```
protected void onCreate(Bundle pSavedInstanceState)
{
   super.onCreate(pSavedInstanceState);
   …
   setLayout();
   …
   initData();
   initUiElements();
   initFragments();
   callServiceOnCreate();
}
```

Listing 5-1: Initialisierungsreihenfolge von JVx Screens

Das Abschicken von Requests wird ebenso gekapselt, wie die Behandlung des Callbacks für Serverantworten.

Android schlägt als Best Practice das Konzept des „Master-Detail-View“ vor. Die Master View stellt dabei eine simple Liste mit den wichtigsten Informationen dar, damit der Benutzer, die Benutzerin die Datensätze identifizieren kann. Nachdem ein Datensatz ausgewählt wurde, öffnet sich die dazugehörige Detailansicht.
Die dritte Ebene wäre dann die Formularansicht um den ausgewählten Datensatz bearbeiten zu können.
Umgelegt auf JVx Anforderungen ergeben sich folgende Activities für die Datendarstellung und Bearbeitung:

- **MasterActivity**
  Diese Activity stellt eine einfache „Top Level View“ dar. Sie repräsentiert eine Liste mit Datensätzen. Nach Auswahl eines Eintrages wird die `DetailActivity` gestartet (dritte View Ebene)

- **MasterComplexActivity**
  Die `MasterComplexActivity` stellt ein Gerüst für komplexe Datenstrukturen zur Verfügung. Sie kann verschachtelte Datensätze auflösen und unterstützt zusätzlich verknüpfte Tabellen. Um die Komplexität aufzulösen wird die „Category View“ verwendet. Hierfür wird durch den gleichen Aufbau ebenfalls die MasterComplexActivity verwendet.
- **DetailActivity**
  Die `DetailActivity` dient der visuellen Darstellung der letzten Ebene verschachtelter Informationen. Laut Android Patterns wird diese gleich der Bearbeitungsansicht gesetzt. Aus Gründen die im Abschnitt 5.3.1 besprochen werden, wird diese am Client von der Bearbeitungsdarstellung entkoppelt.

- **FormActivity**
  Diese Activity ist zuständig um den gewählten Datensatz zu editieren. Sie bildet die unterste Schicht der grafischen Darstellungsmöglichkeiten.

In folgender Abbildung wird der Vergleich mit dem Android Pattern grafisch veranschaulicht.

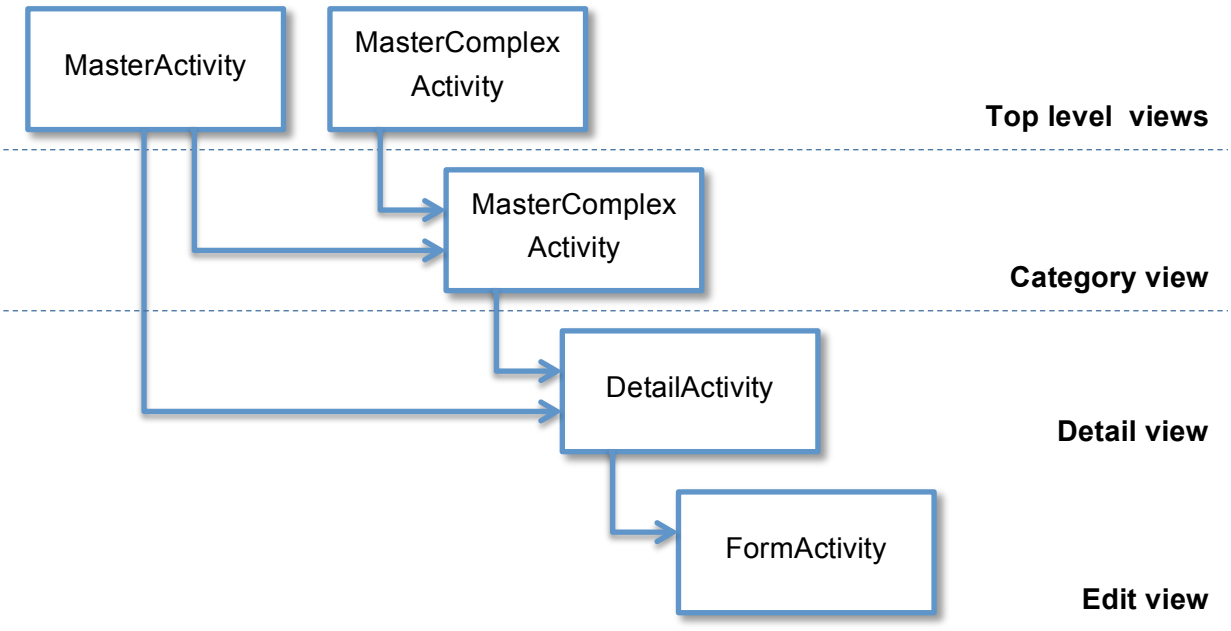

Abbildung 5.2: Android GUI Pattern und Umsetzung im Android Client[28], eigene Überarbeitung

[28] Quelle: http://developer.android.com/design/patterns/app-structure.html, letzter Zugriff 28.3.2014

5.3 Usability von MyERP

Usability ist ein Qualitätsmerkmal. Es beschreibt wie einfach Userinterfaces bedient werden können [42]. Usability wird durch folgende 5 Komponenten definiert [42]:

- **Erlernbarkeit**: Gibt an wie schnell eine Aufgabe von Benutzern, Benutzerinnen bewältigt werden kann, ohne sich davor mit dem Design beschäftigt zu haben.
- **Effizienz**: Spiegelt den Workflow wieder, wie schnell dieser erlernt werden kann um gewünschte Resultate zu erzielen.

- **Wiedererkennung**: Das User Interface mit den Interaktionsmöglichkeiten soll dem Benutzer, der Benutzerin Erinnerung bleiben. Dies kann unterstützt werden durch beispielsweise bekannte Interaktions-Pattern und Verwendung von Standards.

- **Fehler**: Wie viele Fehler werden ausgelöst und wie kann der Anwender, die Anwenderin davon lernen?

- **Zufriedenheit**: Dies spiegelt den Eindruck der durch die Bedienung entsteht wider.

Um die Qualitätskriterien bewerten zu können ist ein ausführlicher Usability-Test notwendig. Dies würde genügend Diskussionsbedarf bieten für eine zusätzliche Master Thesis. Auf Grund dessen wurde in der Arbeit auf bewährte Standards gesetzt und auf den von Android vorgegebenen Guidelines aufgebaut.

5.3.1 Designentscheidungen in Bezug auf die Usability

Wie im vorigen Abschnitt angesprochen, beschränken sich die Designentscheidungen auf Android Standards und basieren auf den Konzepten vorhandener Androidanwendungen wie beispielsweise der Kontakte oder der Mail-Applikation.

Der Benutzer, die Benutzerin soll auf bewährte Interaktionsmuster treffen. Es vereinfacht die Erlernbarkeit der Anwendung. Der Zeitraum zwischen dem ersten Berührungspunkt und dem produktiven Arbeiten wird verkürzt. Durch diese eingesetzten Standards kann die Anwendung effizient genutzt werden.

Gerade bei Datenbankanwendungen kann es vermehrt zu Fehlern kommen. Es werden nur für den Benutzer, der Benutzerin relevante Fehlerursachen angezeigt. Alle anderen werden im Hintergrund behandelt. Die Applikation versucht soweit es möglich ist diese selbst zu beheben durch beispielsweise erneute Requests.

Eine Herausforderung stellte die Tatsache dar, dass Daten bearbeitet werden können. In der Android Kalender Anwendung gelangt man nach Auswahl eines Eintrages sofort zu einer Maske um gegebenenfalls Daten ohne weiterer Interaktion bearbeiten zu können.
Dies stellt bei ERP Applikationen ein Problem dar.
Durch die Verarbeitung heikler Daten muss dem Benutzer, der Benutzerin Sicherheit vermittelt werden. Heikle Operationen wie das Löschen und Speichern von Datensätzen müssen getrennt werden von der Detailanzeige. In der Android Kalender Anwendung öffnet sich nach einem Tap auf einen Kalendereintrag die Detailseite mit der Möglichkeit sofort Daten zu bearbeiten. Um den Anwender, die Anwenderin darin zu unterstützen nicht das Falsche machen zu können wurde eine Übersichtsseite eingeführt mit der Auflistung aller Einzelheiten eines Datensatzes. Erst durch eine weitere Aktion kann dieser gelöscht oder bearbeitet werden.

Durch diese Trennung werden Fehler vermieden indem heikle Operationen expliziert ausgelöst werden müssen. Als weitere Folge erhöht sich die Zufriedenheit.

5.4 Kommunikation mit JVx.mobile

Jede Androidanwendung wird in einem eigenen Prozess ausgeführt. Dabei bekommt das Userinterface einen eigenen Thread wie auch Aufgaben die für die Verarbeitung mehr Zeit benötigen. Dadurch werden Anwender, Anwenderinnen bei der Ausführung blockiert.
Requests an Web-Services im UI-Thread werden standardmäßig von Android bemerkt und führen zu einer Fehlermeldung. Diese Aufgaben müssen in einen eigenen Thread ausgelagert werden der im Hintergrund unabhängig abläuft (Asynchrone Kommunikation).
Um den Response an den UI-Thread weiterzuleiten wird ein Callback-Interface implementiert.

Jede Activity basiert auf der abstrakten Klasse `BaseFragmentActivity`, welche die Initiierungsreihenfolge festlegt und Hooks für die konkreten Klassen zur Verfügung stellt. Die ausimplementierte Callbak-Methode `onRequestFinished` beinhaltet die Logik zum Laden der vom Server geschickten Views. Der Aufbau, sowie die Kommunikation mit dem asynchronen Hintergrundprozess `AsyncTask` wird in folgender Grafik veranschaulicht.

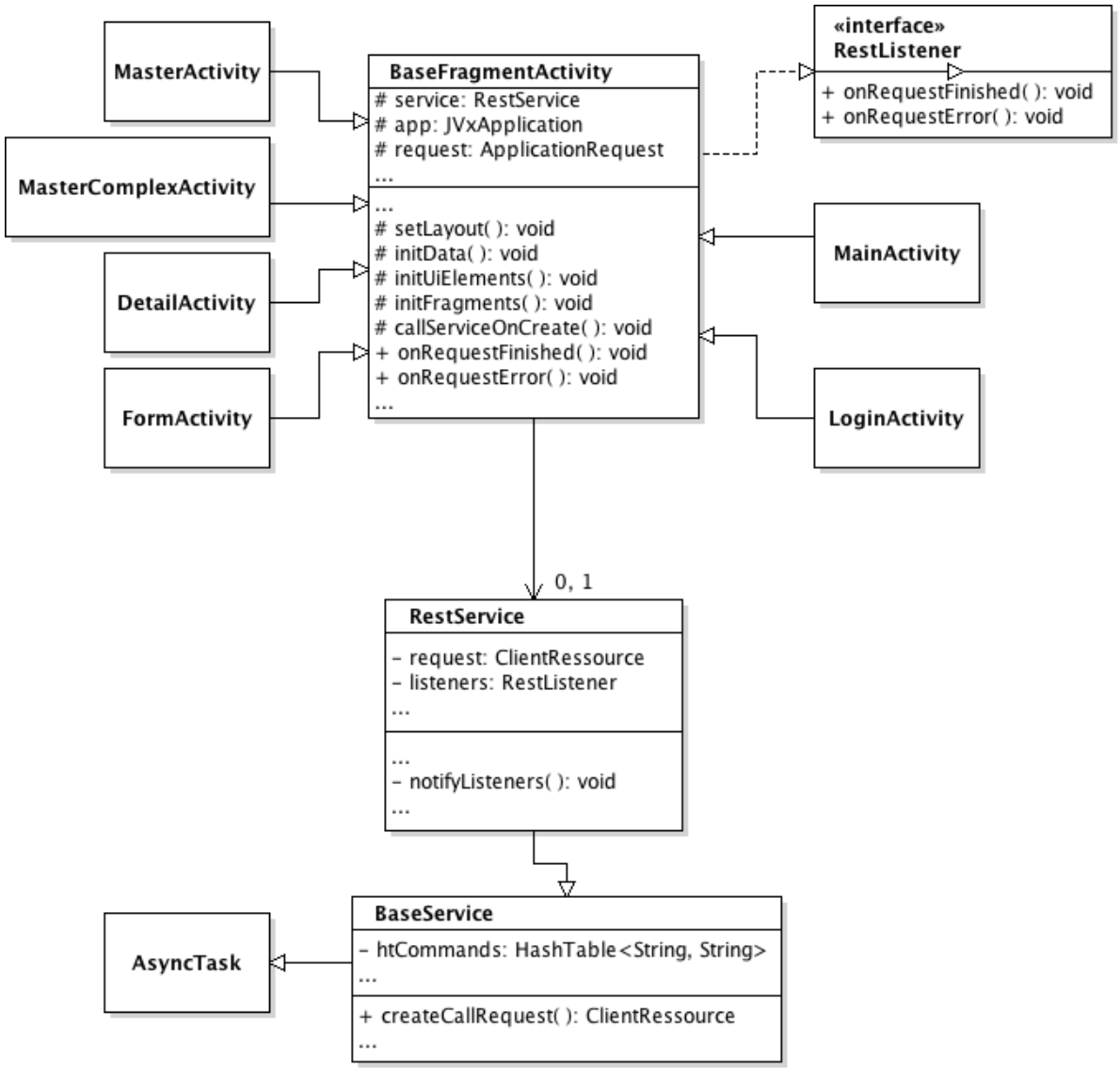

Abbildung 5.3: Klassendiagramm der Activities und Zusammenspiel mit dem RestService

Die technische Umsetzung der Kommunikation mit JVx.mobile wird anhand des Startup Requests näher erläutert.

Wenn der Client gestartet wird, bildet die `MainActivity` die erste View die dem Benutzer, der Benutzerin angezeigt wird. Unter anderem findet hier die Initiierung der remote Anwendung statt. Die Kommunikation basiert, wie JVx.mobile, auf dem Restlet Framework.

Die `MainActivity` implementiert die bereitgestellte Methode `callServiceOnCreate` (siehe Abbildung 5.3). Diese wird sobald der Android Client bereit ist aufgerufen.
In dieser wird der asynchrone Hintergrundprozess initiiert mit folgendem Codestück:

```
service = new RestService(“startup”);
service.addListener(this);

StartupRequest request = new StartupRequest();
request.setApplicationName(app.getApplicationName());
…
```

Listing 5-2: Initiierung des asynchronen Hintergrundprozesses

Der Übergabeparameter „startup“ teilt dem Service den Request-Typ mit. Damit das Userinterface, welches in einem separatem Thread ausgeführt wird, über Ereignisse im Hintergrund-Thread informiert wird und entsprechend reagieren kann, muss zusätzlich der entsprechende `RestListener` gesetzt werden.
Die Client-Parameter werden in einem eigenen Request Objekt gekapselt.
Das Service wird dann letztendlich foglendermaßen gestartet:

```
service.execute(request);
```

Listing 5-3: Starten des REST Services

Während der Ausführung kann der Anwender, die Anwenderin die Applikation weiterhin verwenden ohne blockiert zu werden.

Das `RestService` stellt anhand des Übergabeparameters die entsprechende Client-Ressource zusammen und schickt die Daten in der JSON Repräsentation über die HTTP POST Methode an JVx.mobile (siehe Listing 5-4 a). Hierfür wird auf das Restlet Framework zurückgegriffen. Der erhaltene Response wird vom Framework in die

`Representation`-Klasse umgewandelt, bevor er vom Client in ein Bean-Objekt entpackt wird. Dieses wird danach über die Callback-Methode `notifyListeners` an den GUI-Thread übermittelt (siehe Listing 5-4 b).

```
a) request = createCallRequest(lastRquest, requestType);
   Representation response = request.post(…);
   liResponse = new BeanList<NamedResponse>(response,
   NamedResponse.class);

b) listener.onRequestFinished(liResponse);
```

Listing 5-4: a) Erstellung der Clientressource und Absenden an den Server. b) Retournierung des Server-Respons an den GUI-Thread

Hier übernimmt die ausimplementierte Methode `onRequestFinished` der Klasse `BaseFragmentActivity` die Anzeigelogik. Je nach Response wird beispielsweise das Layout für eine komplexe (`ComplexTableView`) oder einer simplen Datenstruktur (`SimpleTableView`) geladen.

An dem obengenannten Beispiel wird das Konzept der Terminal-Architektur sichtbar. Der Server entscheidet welches Layout für die mitgeschickten Daten am Besten für die Anzeige am Bildschirm geeignet ist. Da diese bereits am Server optimal für die Client-Beschaffenheit aufbereitet wurden, benötigt der Client keine weitere Logik. Dadurch wird eine Applikationsunabhängigkeit gewährleistet.
Durch die Client-Architektur und der zentralen Behandlung von Server-Antworten, können Applikationsentwickler, Applikationsentwicklerinnen diesen einfach erweitern und warten.

6 Conclusio und Ausblick

Die Firma SIB Visions GmbH bietet mit ihrem JVx-Framework eine Möglichkeit für Applikationsentwickler, Entwicklerinnen um hoch performante, anpassbare ERP-Anwendungen zu erstellen. Durch das technologieunabhängige User Interface können diese auf verschiedenen Plattformen deployed werden.

Die Master Thesis schaffte es JVx fit für den mobilen Trend zu machen.
Immer mehr Unternehmen möchten interne Daten „just in time“ abrufen und auswerten können. Smartphones sind allgegenwärtig und immer einsatzbereit. Sie bieten sich an diese Hürde zu meistern. Durch diverse Einschränkungen mobiler Geräte musste im ersten Schritt in Server implementiert werden. JVx.mobile stellt die Anwendung als entfernte Benutzerschnittstelle zur Verfügung. Er hat die Aufgabe Clientanfragen auszuwerten, die gewünschten Daten zu sammeln und sie in einer optimierten Version als Response zurück zu schicken. Die Kommunikation läuft dabei über ein REST Service. Durch das obengenannte Konzept ist es möglich auch in Zukunft weitere mobile Endgeräte in JVx zu implementieren.

Die Herausforderung dabei lag bei der Datenaufbereitung. Jeder Client hat andere Constraints die berücksichtigt werden müssen.
Um die Vorteile eines Frameworks nutzen zu können, wurde die komplette Geschäfts- und Darstellungslogik auf den Server ausgelagert. Bei der Authentifizierung können Clientparameter mitgeschickt werden, die für die Analyse herangezogen werden. Somit bekommt jeder Client personalisierte Daten.

Das mobile Endgerät stellt eine grafische Oberfläche zur Verfügung um die entfernte Anwendung steuern zu können. Durch die Entkoppelung stellt der entwickelte Android Client ein Terminal dar. Die Hauptaufgabe ist es für eine benutzerfreundliche Oberfläche zu sorgen sowie die Weiterleitung der Useraktionen an den Server.
Der Wartungsaufwand bei Änderungen oder Adaptionen sowie Erweiterungen wird dadurch verringert.

Der Trend in Richtung Mobilität wird immer bedeutsamer und schafft neue Produkte und Dienstleistungen am Markt. JVx kann nun mit der neu entwickelten Funktionalität im mobilen Sektor mitspielen und bietet als einer der wenigen ein „Full-Stack-Framework“.

Da die Datenflut an Informationen immer weiter steigt, sollte auch JVx.mobile reagieren. Im Kapitel 4.4.3 wurden Optimierungsansätze beschrieben.

Literaturverzeichnis

[1] A. Alkassar, M. Garschhammer, F. Gehring, P. Keil, H. Kelter, U. Löwer, M. Pankow, A.R. Sadeghi, M. Schiffers, M. Ullmann, S. Vogel, Kommunikations- und Informationstechnik 2010+3: Neue Trends und Entwicklungen in Technologien, Anwendungen und Sicherheit [Online] Verfügbar: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Trend2010/summary_pdf.pdf?__blob=publicationFile (letzter Zugriff: 2.4.2014)

[2] E.M. Shehab, M.W. Sharp, L. Supramaniam, T.A. Spedding, Enterprise resource planningAn integrative review, Business Process Management Journal Vol. 10 No. 4, 2004 pp. 359-386

[3] N. Gronau, Enterprise Resource Planning – Architektur, Funktionen und Management von ERP-Systemen, 2. Auflage, Oldenbourg 2010

[4] E. F. Monk, W. J. Wagner, Concepts in Enterprise Resource Planning, 4. Auflage, Course Technology, USA 2013

[5] J. G. Harris, T. H. Davenport, New Growth from Enterprise Systems: Achieving High Performance through Distinctive Capabilities, Accenture 2006

[6] E.M. Shehab, M.W. Sharp, L. Supramaniam, T.A. Spedding, Enterprise resource planning – An integratice review, Business Process Management Journal Vol. 10 No. 4, 2004, pp. 359-386

[7] Panorama Consulting Solutions, 2013 ERP RRPORT [Online] Verfügbar: http://go.panorama-consulting.com/rs/panoramaconsulting/images/2013-ERP-Report.pdf (letzter Zugriff: 2.4.2014)

[8] J. Helmle, mySAP ERP im Kontext der SAP Produktstrategie, SAP Deutschland AG & Co. KG [Online] Verfügbar: http://ginnold.de/_old/stuff/sapat/at19/helmle.pdf (letzter Zugriff: 2.4.2014)

[9] A. Bonaccorsi, C. Rossi, *Why Open Source software can succeed*, Laboratory of Economics and Management, Sant‘ Anna School of Advanced Studies, Italy 2003

[10] Open Source Initiative [Online] Verfügbar: http://opensource.org (letzter Zugriff: 1.4.2014)

[11] Free Software Foundation, Kategorien freier und unfreier Software [Online] Verfügbar: http://www.gnu.org/philosophy/categories.de.html#OpenSource (letzter Zugriff: 1.4.2014)

[12] T. Rosenkranz, Open Contents, Mohr Siebeck Tübingen, 2011

[13] M. Hansen, K. Köhntopp, A. Pfitzmann, The Open Source approach - opportunities and limitations with respect to security and privacy, Elsevier Science, 2002

[14] B. Johansson, F. Sudzina, ERP systems and open source: an initial review and some implications for SMEs, Journal of Enterprise Information Management Vol. 21 No. 6, 2008, pp. 649-658, Emerald Group, 2008

[15] A. MacCormack, J. Rusnak, C. Baldwin, Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code, Management Science, 2006

[16] K. Pousttchi, B. Thurnher, Usage of mobile technologies to support business processes, University of Augsburg, 2006

[17] M. Scherz, Mobile Business: Schaffung eines Bewusstseins für mobile Potenziale im Geschäftsprozesskontext, Technische Universität Berlin, 2008

[18] M. Kakihara, C. Sorensen, Mobility: An Extended Perspective, Department of Information Systems, Proceedings oft he 35th Hawaii International Conference on System Sciences, 2002

[19] A. Tanenbaum, M. Stehen, Verteilte Systeme Prinzipien und Paradigmen, 2. Aktualisierte Auflage, Pearson Studium, 2008

[20] P.A. Bernstein, Middleware: a model for distributed system services, Magazine Communications of the ACM, Volume 39, Pages 86-98, New York 1996

[21] S.Wurm, Umsetzung einer auf Java EE 6 basierenden generischen Persistenzumgebung für das quelloffene Framework JVx, Wien 2012

[22] Development, SIB Visions 2010 [Online] Verfügbar: http://forum.sibvisions.com/viewtopic.php?f=5&t=75 (letzter Zugriff: 14.4.2014)

[23] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns – Elements of Reusable Object-Oriented Software, ADDISON-WESLEY, 1995

[24] W. Codenie, K.D. Hondt, P. Steyaert, A. Vercammen, FROM CUSTOM APPLICATIONS TO DOMAIN-SPECIFIC FRAMEWORKS, Communications of the ACM, Voll. 40 No. 10, 1997

[25] G. Froehlich, H.J. Hoover, L. Liu, P. Sorenson, Hooking into Object-Oriented Application Frameworks, Department of Computing Science, Alberta, 1997

[26] M.E. Fayad, D.C. Schmidt, Object-Oriented Application Frameworks, Communications of the ACM, Vol. 40 No. 10, 1997

[27] R.E. Johnson, FRAMEWORKS = (COMPONENTS + PATTERNS), Communications of the ACM, Vol. 40 No 10, 1997

[28] SIB Visions [Online] Verfügbar: http://www.sibvisions.com (letzter Zugriff: 20.4.2014)

[29] M. Knutz, Web Services – Einführung und Übersicht, Reihe Javamagazin, Software & Support Verlag, Frankfurt 2002

[30] World Wide Web Consortium [Online] Verfügbar: http://www.w3.org/, (letzter Zugriff: 20.4.2014)

[31] D. Kossmann, F. Leymann, Web Services, Springer Verlag, 2004

[32] L. Richardson, S. Ruby, RESTful Web Services, O'Reilly Media, 2007

[33] J. Louvel, T. Templier, T. Boileau, Restlet IN ACTION, Manning Publications Co, 2013

[34] L. Richardson, S. Ruby, Web Services mit REST, O'Reilly, Köln 2007

[35] Restlet, Restlet API [Online] Verfügbar: http://restlet.org/learn/javadocs/2.2/jee/, (letzter Zugriff: 26.4.2014)

[36] Google, Platform Versions [Online] Verfügbar: http://developer.android.com/about/dashboards/index.html (letzter Zugriff: 19.4.2014)

[37] SIB Visions, JVx - Enterprise Application Framework (ver. 1.2) [Online] Verfügbar: http://www.sibvisions.com/files/jvx/1.2/api/ (letzter Zugriff: 19.4.2014)

[38] M. Hofer, Javadoc JVx.mobile, Verfügbar auf der Begleit-CD

[39] M. Fowler, Patterns of Enterprise Application Architecture, Addison Wesley, 2002

[40] Google, Support Library [Online] Verfügbar: http://developer.android.com/tools/support-library/index.html (letzter Zugriff 26.4.2014)

[41] R. Meier, Professional – Android™ 4 Application Development, John Wiley & Sons, Indianapolis, 2012

[42] J. Nielsen, Usability 101: Introduction to Usability, Nielsen Norman Group, 2012

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

API	Application Programming Interface
CRM	Customer Relationship Management
ERP	Enterprise Resource Planning
HRM	Human Relationship Management
HTTP	HyperText Transfer Protokoll
MRP	Material Requirement Planning
MRP II	Manufacturing Resource Planning
OSS	Open source Software
REST	Representation State Transfer
ROP	Reorder Point
WS	Web Service