AMPERE AG
ENERGIE-BROKER

Freie Universität Berlin

Fachbereich: Mathematik und Informatik
Institut für Informatik
Arbeitsgruppe: Software Engineering

Bachelor Thesis

zur Erlangung des Grades Bachelor of Science

Thema

Implementierung einer Webservice-Schnittstelle für die Dokumentengenerierung mit JasperReports

vorgelegt von Mohamed Keita (keita@inf.fu-berlin.de, Matrikelnummer: 4128753)
Betreuer und Erstgutachter: Prof. Dr. Lutz Prechelt
Zweitgutachter: Prof. Dr. Elfriede Fehr

Berlin, 7. Juni 2012

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Bücher, Internetquellen oder ähnliches sind im Literaturverzeichnis angegeben. Zitate aus fremden Arbeiten sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, 7. Juni 2012

Mohamed Keita

Vorwort und Danksagung

Die Idee für diese Bachelorarbeit entstand im Juli 2011 bei der Firma Ampere AG[1], der ich für die Ermöglichung und die Unterstützung dieser Arbeit danken möchte. Das Thema wurde in einem produktiven Gespräch mit meinem Teamleiter Herrn Marco Keeren (Ampere AG) und meinem Projektleiter Herrn Nico Awe (Ampere AG) durchgesprochen und entwickelt. Ich möchte mich an der Stelle bei beiden ebenfalls für ihre Unterstützung bedanken. Mein Dank geht selbstverständlich an das Team Pool-Management für das Testen der Software und Feedback.

Ich bedanke mich ganz besonders bei Herrn Prof. Dr. Lutz Prechelt[2] für die Betreuung dieser Bachelorarbeit.

1 Das Unternehmen – Ampere AG: http://ampere.de/das-unternehmen.php (Abruf: 18.05.2012)

2 Freie Universität Berlin - Professor of Informatics and head of the software engineering research group (AG SE). http://www.mi.fu-berlin.de/w/Main/LutzPrechelt (Abruf: 18.05.2012)

Abstract

Die Ampere AG betreut als Energie-Dienstleister über 14.000 gewerbliche Strom- und Gaskunden in Deutschland. Die Ampere AG schafft einen Überblick über die Energienachfrage ihrer Kunden beim Strom- und Gaseinkauf und nimmt Preisverhandlungen mit Energieversorgern für sie auf. Durch ihre Dienstleistungen entlastet die Ampere AG ihre Kunden von der aufwändigen Suche nach dem besten Angebot bzw. ermöglicht ihnen den Wechsel zu einem günstigeren Energieversorger. Die zunehmende Komplexität oben genannter Dienstleistungen der Ampere AG bei der Kommunikation mit ihren Kunden erfordert ein effizienteres Dokumentengenerierungssystem. Dieses System wurde im Rahmen dieser Bachelorarbeit realisiert.

Die vorliegende Arbeit stellt Vorgehen, Methoden, Technologien und Werkzeuge vor:

- für die Entwicklung einer SOAP-basierten Webservice-Schnittschnittstelle, welche eine bereits vorhandene Datenbankapplikation um eine weitere Komponente für die Dokumentengenerierung mit der JasperReports Library erweitert.

- für die Umsetzung der Dokumentenvorlagen des Unternehmens unter Einsatz vom iReport-Designer.

Der Webservice soll dabei dem Anwender die Möglichkeit geben, aus der Firmendatenbank Dokumente in unterschiedlichen Formaten (z. B. PDF, DOCX, CSV) zu generieren. Ein Dialogfenster nimmt die vom Anwender definierten Parameter entgegen und übermittelt diese inklusiv der im Dokument auszugebenden Daten als SOAP-Request an den Webservice. Durch den Einsatz des Webservices wird der Arbeitsaufwand der Anwender im Unternehmen wesentlich verringert. Per Knopfdruck erhält der Anwender das fertige Dokument im gewünschten Format.

Inhaltsverzeichnis

Eidesstattliche Erklärung 2
Vorwort und Danksagung 3
Abstract 4
1 Einführung 6
1.1 Aufgabenstellung und Zielsetzung 6
1.2 Vorgehen 7
1.3 Aufbau der Bachelorarbeit 8
2 Globale Architekturentscheidung 9
2.1 Service Oriented Architecture (SOA) 9
2.2 Webservices 10
2.3 SOAP 11
2.4 WSDL 12
3 Methoden 14
4 Werkzeuge und Technologien 15
4.1 Werkzeuge 15
4.2 Technologien 16
5 Analyse von iReport-Designer und JasperReports 19
5.1 Funktionalitäten 19
5.2 Funktionsmängel 21
5.3 Vorteile für ein Unternehmen 22
5.4 Vorteile für den Entwickler 23
6 Umsetzung des Webservices 24
6.1 Architektur 25
6.2 Aufbau des Softwareprojektes 26
6.2.1 JasperDeveloper-Projekt 27
6.2.2 JasperWS-Projekt 28
6.3 Webservice - Ordnerstruktur 29
6.4 Webservice – Klassenübersicht 31
6.5 JasperReports API Overview 33
6.6 Komponenten des Webservices 34
6.7 Aufruf des Webservices 35
6.8 Aktivitätsdiagramm 38
7 Umsetzung der Dokumentenvorlagen 39
7.1 Begriffe und Abkürzungen 39
7.2 Ausgangssituation 39
7.3 Musterdesign 40
7.4 Vertragsschreiben 41
7.5 Kundenreport 43
8 Fazit 45
Anhang 47
Anhangverzeichnis 47
Abbildungsverzeichnis 54
Quelltextverzeichnis 61
Literaturverzeichnis 76

1 Einführung

In diesem einleitenden Kapitel werden zunächst in die Aufgabenstellung und die dahinter stehenden Ziele der Bachelorarbeit eingeführt. Des Weiteren werden die planmäßige Vorgehensweise zum Erreichen des gesetzten Ziels sowie der Aufbau der vorliegende Bachelorarbeit erläutert.

1.1 Aufgabenstellung und Zielsetzung

Die Ampere AG setzt als Energiebroker bei der Kommunikation mit ihren Kunden eine Vielzahl von unterschiedlichen Schreiben ein. Dabei kann es sich beispielsweise um Informationsschreiben zum aktuellen Vertragsstatus, Honorarrechnungen oder komplexe Auswertungen (Analysen) der Abnahmestellen eines Kunden handeln. Bisher wurde dazu für jeden Dokumenttyp eine statische Vorlage in MS-Word angelegt, welche durch Feldfunktionen mit Kundendaten gefüllt wurde. Die zunehmende Komplexität oben genannter Servicedienstleistungen der Ampere AG erfordert ein flexibleres und effizienteres Dokumentengenerierungssystem. Dieses System soll nun mit der vorliegenden Arbeit realisiert werden.

Die Ampere AG ist als Energiedienstleister stets bemüht, seinen Kunden eine effiziente Dienstleistung anzubieten. Das Ziel dieser Bachelorarbeit ist es, eine effiziente Softwarelösung zu entwickeln, welche das Unternehmen dabei unterstützt seine Arbeitsprozesse effektiver zu organisieren und damit für einen schnellen und reibungslosen Service für seine Kunden zu sorgen. In Zukunft soll daher die Generierung von Dokumenten mit Hilfe einer Webservice-Schnittstelle realisiert werden.

1.2 Vorgehen

Zum Erreichen des festgesetzten Zieles spielt eine planmäßige Vorgehensweise bei einer erfolgreichen Software-Entwicklung eine sehr große Rolle. Dabei soll die Softwaretechnik in ihren Aufgaben den Beteiligten (Kunde, Benutzer, Manager und Entwickler) helfen, folgende softwaretechnische Fragen zu klären:

- Kosten/Nutzen-Analyse
- Anforderungsanalyse
- Entwurf
- Implementierung
- Qualitätssicherung und -management.

Zunächst wurden die Dokumentenvorlagen des Unternehmens auf ihre Umsetzbarkeit geprüft. Im Kapitel 5 werden die Aspekte sowie die Ergebnisse der Analyse vorgestellt. Im Anschluss an die Analyse wurde in der IT-Abteilung des Unternehmens über die Datenmodelle des zu entwickelnden Webservices intensiv diskutiert. Die gemeinsam festgelegten Datentypen dienten als Grundlage für die Erarbeitung eines Lösungsdesigns, welches die Basis für die Entwicklung eines effizienten Dokumentengenerierungssystems bildete. Im Kapitel 6 wird das Dokumentengenerierungssystem ausführlich vorgestellt.

1.3 Aufbau der Bachelorarbeit

Im Anschluss an diese Einführung werden im Kapitel 2 die grundlegende Architektur und die Begriffe beschrieben, welche die Basis für das Verständnis der, in dieser Bachelorarbeit vorgestellten, Softwarelösung sind.

Die Kapitel

Das nachfolgende Kapitel 3 und 4 geben einen detaillierten Überblick über die Methoden, Werkzeuge und Technologien, welche bei der Lösung des, in der Aufgabenstellung bereits vorgestellten, Problems eingesetzt wurden.

Im Kapitel 5 werden die Werkzeuge iReport-Designer und der JasperReports auf ihre Einsetzbarkeit in Rahmen dieses Projektes untersucht. Vor- und Nachteile aus Unternehmer- und Entwicklersicht werden dargestellt.

Das Kapitel 6 stellt die Umsetzung der Softwarelösung für die Dokumentengenerierung vor. Die praktische Umsetzung von Dokumentenvorlagen für die Ampere AG findet sich im Kapitel 7.

Das Fazit in Kapitel 8 enthält eine kurze Beschreibung der praktischen Anwendung des Systems bei der Ampere AG.

Quelltexte, ergänzende Abbildungen sowie das Literaturverzeichnis und das Abbildungsverzeichnis befinden sich in Anhang.

2 Globale Architekturentscheidung

Dieses Kapitel befasst sich mit der Softwarearchitektur, Technologien und Standards, welche die Grundlagen bei der Umsetzung eines Dokumentengenerierungssystems bilden. Dabei soll zunächst einen Einblick in das Paradigma Service Oriented Architecture (kurz SOA) sowie in das Thema Webservice gewonnen werden.

2.1 Service Oriented Architecture (SOA)

SOA (Service Oriented Architecture) ist ein Konzept zur Bereitstellung von unabhängigen und wiederverwendbare Diensten. Dabei ist es wichtig, dass Servicebeschreibung und Serviceimplementierung getrennt sind.

- **Servicebeschreibung:** Hier wird definiert, welche Operationen ein Service anbietet und wie sie genutzt werden können. Für den Aufrufer sind Details der Implementierung völlig irrelevant. Er muss weder die verwendete Architektur noch die zugrunde liegende Plattform kennen. Für ihn ist nur die definierte Schnittstelle von Bedeutung.

- **Serviceimplementierung:** Hier erfolgt die konkrete Umsetzung der Service-Schnittstelle.

Die Service orientierte Architektur basiert auf der Interaktion zwischen drei Rollen:

- **Service Provider:** implementiert einen Service und stellt ihn zur Verfügung. Der Service besitzt eine klar definierte Schnittstelle.

- **Service Broker:** fungiert als Vermittler zwischen Service Provider und Service Consumer.

- **Service Consumer:** ist der Nutzer eines Services. Das kann beispielsweise eine eine Person sein, die über einem Browser ein Webservice aufruft oder auch eine Datenbankapplikation, welche einen Service aufruft.

2.2 Webservices

Webservices sind eine Technologie für eine plattformunabhängige Kommunikation zwischen Computern über ein Netzwerk.

Ein Anbieter stellt bestimmte Funktionen zur Verfügung, die von einem oder mehreren Nutzern in Anspruch genommen werden. Die Kommunikation zwischen dem Anbieter der Services und deren Nutzern erfolgt über Nachrichten, die auf XML basieren und meist mittels HTTP übertragen werden.

Eine detailierte Definiton findet sich in dem Online Tutorial der Softwarefimea Oracle „The Java EE 6 Tutorial":

"What Are Web Services?

Web services are client and server applications that communicate over the World Wide Web's (WWW)HyperText Transfer Protocol (HTTP). As described by the World Wide Web Consortium (W3C), web services provide a standard means of interoperating between software applications running on a variety of platforms and frameworks. Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions, thanks to the use of XML. Web services can be combined in a loosely coupled way to achieve complex operations. Programs providing.

Types of Web Services

On the conceptual level, a service is a software component provided through a network-accessible endpoint. The service consumer and provider use messages to exchange invocation request and response information in the form of self-containing documents that make very few assumptions about the technological capabilities of the receiver. 359 On a technical level, web services can be implemented in various ways. The two types of web services discussed in this section can be distinguished as "big" web services and "RESTful" web services"[3]

3
The Java EE 6 Tutorial. Copyright © 2012, Oracle and/or its affiliates. All rights reserved. PartNo: 821–1841–15 April 2012. http://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf, Abruf: 03.06.2012

2.3 **SOAP**

SOAP steht für Simple Object Access Protocol. Es ist ein XML-basiertes Protokoll für den Austausch strukturierter Informationen in einer dezentralisierten, verteilten Umgebung. SOAP ist unter W3C[4] spezifiziert und standardisiert. Es wird dort wie folgt beschrieben:

"SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework."[5]

4 W3C Note 08 May 2000
W3C: http://www.w3.org/TR/soap/ (Abruf: 18.05.2012)

5 W3C Note 08 May 2000
SOAP. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ 89: (Abruf: 18.05.2012)

2.4 **WSDL**

WSDL steht für Web Service Description Language. Mit WSDL wird die Schnittstellendefinition einer XML basierten Web Service beschrieben. Dabei werden die Methoden (mit deren Ein-/Ausgabendatentypen), die aufgerufen werden können, definiert. Des Weiteren wird definiert, wo der Service zu finden ist und wie er angesprochen wird.

- Eine weitere Defintion von der W3C:

„WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME."[6]

6 W3C Note 15 March 2001 http://www.w3.org/TR/wsdl.(Abruf: 21.05.2012)

- **WSDL Syntax**

Ein WSDL Dokument besteht aus dem Wurzelelement (definitions) und fünf weitere Elementen: *types*, *message*, *portType*, *binding* und *service*. Die Fünf Elemente beschreiben die verschiedenen Aspekte eines Web Service.

Die Elemente *types*, *message* und *portType* bilden den abstrakten Teil des Webservices. Dieser Teil beschreibt die Schnittstelle des Service (z. B. Operationen, Ein-/Ausgabedatentypen) unabhängig vom Transportprotokoll und der konkreten Implementierung. Der Hauptteil (*binding und service*) beschreiben, wie der Webservice erreichbar ist (URI, Protokoll) und wie die Daten serialisiert und codiert werden.

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

<wsdl:definitions

        <wsdl:types></wsdl:types>

        <wsdl:message>...</wsdl:message>

        <wsdl:portType>...</wsdl:portType>

        <wsdl:binding>...</wsdl:binding>

        <wsdl:service>...</wsdl:service>

</wsdl:definitions>
```

3 Methoden

Bei der Implementierung der Webservice-Schnittstelle und der Umsetzung der Dokumentenvorlagen wurden folgende Methoden verwendet:

- Bottom-Up-Design

„Beim Bottom-Up-Design (Basis-Ansatz) werden zunächst einzelne Programmbestandteile (z. B. Funktionen, Klassen, Module) definiert, eventuell sogar direkt geschrieben. Aus diesen Teilen werden dann größere Elemente des Programms zusammengesetzt, bis das vollständige System erstellt ist. Dabei sollte darauf geachtet werden, dass ein Quelltext entsteht, den man jederzeit in einem anderen Projekt verwenden kann"[7]. Unter Einsatz oben genannter Methode lässt sich die Webservice-Schnittstelle mit Hilfe vom „Apache Axis Emitter Framework" einfach definieren.

- Master-/Subreports: Ein zentraler Aspekt des Konzeptes Master-/Subreport[8] ist die Wiederverwendbarkeit von „JasperReports Templates" (ein Subreport kann in verschiedenen Masterreports wiederverwendet werden). Dies ist beispielsweise der Fall bei der Umsetzung von komplexeren und aufeinander aufbauenden Reports.

- Main-/Subdatasets[9]

- Bands

- Groups

- File Virtualizer

---

7 Bottom-Up-Design: http://de.wikipedia.org/wiki/Top-Down-_und_Bottom-Up-Design (Abruf: 21.05.2012)

8 Konzept Master-/Subreports: [1], CHAPTER 11: Subreports, S.129-135. [2], CHAPTER 6: Report Layout and Design. Subreports, S.162-168. [3], CHAPTER 6: Report Layout and Design. Subreports, S.173-179. [4], CHAPTER: Subreports, S.161-168. [5], CHAPTER 10: Subreports, S.153-171. [6], CHAPTER 8: Subreports, S.109-125.

9 Konzept Main-/Subdatasets: [1], CHAPTER 12: Datasets, S.179. [2], CHAPTER: Subdatasets, S.205-212. [3], Chapter 8: Other JasperReports Features. Subdatasets, S.222-229. [4], CHAPTER: Datasets, S.169-171. [5], CHAPTER 12.2: Using Datasets, S.228-235. [6], CHAPTER 14: Subdatasets, S.221-231.

4 Werkzeuge und Technologien

In diesem Abschnitt werden die Werkzeuge sowie die Technologien, welche bei der Entwicklung der Webservice-Schnittstelle für die Dokumentengenerierung eingesetzt wurden, vorgestellt.

4.1 Werkzeuge

- Eclipse IDE for Java EE Developers
  - Die IDE (engl. Integrated Development Environment)[10] umfasst viele nützliche Features zur Unterstützung eines optimalen Entwicklungsprozesses:
  - Mit dem Subversion-Plugin „Subclipse"[11] lassen sich beispielsweise Quelltexte und weitere nützliche Dateien des gesamten Softwareprojektes einfach verwalten.
  - Eine weitere wichtige Feature der IDE ist der Web Service Explorer. Mit Hilfe des Web Service Explorers lässt sich ein SOAP-basierter Webservice einfach testen. Der Web Service Explorer unterstützt vollständig SOAP Syntax.
- Ireport-Designer: Mit dem iReport-Designer werden Berichtvorlagen erstellt.
- Dia-UML (Unified Modeling Language)
- Eclipse Web Service Explorer
  - Mit dem integrierten Web Service Explorer der „Eclipse IDE for Java EE Developers" wird der Entwickler dabei unterstützt, SOAP-basierte Webservices zu testen, ohne sich mit der Funktionsweise von SOAP befassen zu müssen. Er unterstützt vollständig die „SOAP XML Syntax"[12].

---

10 Eclipse IDE for Java EE Developers: http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/heliossr2 (Abruf: 18.05.2012)

11 Subclipse: http://subclipse.tigris.org/update_1.6.x/ (Abruf; 21.05.2012)

12 SOAP - Spezifikation: http://www.w3.org/TR/soap/ (Abruf: 21.05.2012)

4.2 Technologien

- J2SDK: Java 2 Standard Development Kit
- J2EE: Java 2 Enterprise Edition
- JRE: Java Runtime Environment
- Apache Tomcat Server
- Apache Axis Emiter
  - Das in Eclipse integrierte Framework unterstützt bei der Entwicklung eines SOAP-basierten Webservice. Apache Axis Emitter generiert vollautomatisch die WSDL (Web Service Description Language) und die dazu gehörigen WSDD (Web Services Deployment Descriptor) für die Bereitstellung auf einem Applikationsserver (z. B. Tomcat Server). Mit den integrierten Tools JAVA2WSDL und WSDL2JAVA wird der Entwickler dabei unterstützt, automatisch eine robuste Webservice-Schnittstelle in Java zu erzeugen, ohne sich direkt mit der Funktionsweise von SOAP befassen zu müssen.
- Apache Library
- JasperReports Library: Die Java Bibliothek für die Dokumentengenerierung
- XML: Extensible Markup Language

- XPath 2.0 (XML Path Language)

  - *„XPath 2.0 is an expression language that allows the processing of values conforming to the data model defined in [XQuery 1.0 and XPath 2.0 Data Model (Second Edition)]. The data model provides a tree representation of XML documents as well as atomic values such as integers, strings, and booleans, and sequences that may contain both references to nodes in an XML document and atomic values. The result of an XPath expression may be a selection of nodes from the input documents, or an atomic value, or more generally, any sequence allowed by the data model. The name of the language derives from its most distinctive feature, the path expression, which provides a means of hierarchic addressing of the nodes in an XML tree. XPath 2.0 is a superset of [XPath 1.0], with the added capability to support a richer set of data types, and to take advantage of the type information that becomes available when documents are validated using XML Schema. A backwards compatibility mode is provided to ensure that nearly all XPath 1.0 expressions continue to deliver the same result with XPath 2.0; exceptions to this policy are noted in [**I Backwards Compatibility with XPath 1.0**].“*[13]

13 Xpath: http://www.w3.org/TR/2010/REC-xpath20-20101214/ (Abruf: 21.05.2012)

- JAX-WS

  - *„Java API for XML Web Services (JAX-WS) is a technology for building web services and clients that communicate using XML. JAX-WS allows developers to write message-oriented as well as Remote Procedure Call-oriented (RPC-oriented) web services.*

  - *In JAX-WS, a web service operation invocation is represented by an XML-based protocol, such as SOAP. The SOAP specification defines the envelope structure, encoding rules, and conventions for representing web service invocations and responses. These calls and responses are transmitted as SOAP messages (XML files) over HTTP.*

  - *Although SOAP messages are complex, the JAX-WS API hides this complexity from the application developer. On the server side, the developer specifies the web service operations by defining methods in an interface written in the Java programming language. The developer also codes one or more classes that implement those methods. Client programs are also easy to code. A client creates a proxy (a local object representing the service) and then simply invokes methods on the proxy. With JAX-WS, the developer does not generate or parse SOAP messages. It is the JAX-WS runtime system that converts the API calls and responses to and from SOAP messages.*

  - *With JAX-WS, clients and web services have a big advantage: the platform independence of the Java programming language. In addition, JAX-WS is not restrictive: A JAX-WS client can access a web service that is not running on the Java platform, and vice versa. This flexibility is possible because JAX-WS uses technologies defined by the W3C: HTTP, SOAP, and WSDL. WSDL specifies an XML format for describing a service as a set of endpoints operating on messages."*[14]

14 Oracle: Chapter 19: Building Web Services with JAX-WS

5 Analyse von iReport-Designer und JasperReports

5.1 Funktionalitäten

- Mit dem iReport-Designer lassen sich Design-/Layout- und Stilvorlagen einfach erstellen.
  - Die Vorlagen können sowohl für einzelne Dokumente als auch für Seriendokumente erstellt werden (hierfür werden mehrere Dokumente in einem Schritt generiert).
- Das Hintergrundbild für ein Dokument lässt sich einfach und schnell in die Vorlage einfügen.
- Tabellen können mit unterschiedlichen Stilen erstellt werden.
- TrueType Fonts (TTF's) können in iReport-Designer eingebunden werden.
  - Mit Hilfe der integrierten Exportfunktion können die TTF's einfach als Java Archiv (kurz JAR) exportiert werden.
- Der iReport-Designer unterstützt vollständig JasperReports XML Syntax (kurz JRXML).
  - Die generierte JRXML Datei kann einfach im XML-Modus bearbeitet werden.

- Der Zugriff auf Reportdaten kann auf verschiedenen Wegen realisiert werden:
  - Über eine Datenbank: Mittels eines JDBC-Treibers kann auf die Datenbank zugegriffen werden und die SQL Statements können ausgeführt werden.
  - Per XML: Über einen JRXpathQueryExecuter können Xpath-Expressions ausgeführt werden.
- Die Reportelemente (Text, Frame, Chart, List etc.) können einfach per „Drag and Drop" in die Report-Sections eingefügt werden.
- Mit dem iReport-Designer können unterschiedlichen Barcodes generiert werden.
- Mit dem iReport-Designer können unterschiedlichen Charts dargestellt werden:
  - Pie Charts
  - XY Charts
  - Line Charts
  - Bar Charts
  - Stacked Bar Charts.
- Einer der wichtigen Vorteile von iReport-Designer ist die Unterstützung vom JasperReports-Compiler und -Exporter.
  - Mit den integrierten Compiler und Exporter lassen sich „JasperReports Templates" (JRXML) einfach kompilieren und in verschiedene Formaten exportieren (z. B. PDF, DOCX, CSV).

5.2 Funktionsmängel

Der iReport-Designer hat sich zwar in seinem Funktionsumfang in vielen Bereichen sehr gut bewährt, jedoch hat sich an manchen Stellen der Analyse der Dokumentenvorlagen herausgestellt dass:

- die Vorlagen mit unterschiedlichen Orientierungen (z. B. Hoch- oder Querformat) nicht ohne Weiteres mit dem iReport-Designer angepasst werden können. Zum Zeitpunkt der Generierung des gesamten Reports können Subreports nicht an den Master angepasst werden. Verweist ein Master beispielsweise mit der Orientierung Hochformat auf ein Subreport mit der Orientierung Querformat, welcher eine Tabelle enthält, so führt diese bei Generierung des gesamten Reports zu einem Informationsverlust. Ein Großteil der Tabelle wird einfach abgeschnitten.

- Die Diagramme, die auf einer Pivot-Tabelle[15] basieren, können nicht ohne zusätzlichen Programmieraufwand, dargestellt werden.

  - Das EDV-Programm MS-Excel ist in dieser Hinsicht sehr nützlich. Mit MS-Excel lässt sich eine Pivot-Tabelle erstellen und das Diagramm dazu mit einer sehr hohen Qualität zeichnen.

  - Um dieses Problem zu umgehen, bietet JasperReports eine weitere Bibliothek „JfreeChart"[16].

15 Pivot-Tabelle: http://de.wikipedia.org/wiki/Pivot-Tabelle (Abruf: 19.05.2012)

16 JFreeChart-Bibliothek : http://www.jfree.org/jfreechart/ (Abruf: 19.05.2012)

5.3 Vorteile für ein Unternehmen

- Die JasperReports Library ist Open Source (kostenlos) und sehr weit verbreitet (große Community + Support).

- Mit der JasperReports Library ist die Dokumentengenerierung schneller und effizienter (Per Knopfdruck werden aus komplexeren Datensätze Dokumente generiert.):

  - Der Generierungsprozess ist für den Anwender komplett unsichtbar. Das bedeutet, dass er nur das fertige Dokument im gewünschten Format erhält.

- Die JasperReports Library bietet mehr Flexibilität:

  - Die Dokumentformate sowie die Reportmodule können von dem Anwender selbst über eine Datenbankanwendung ausgewählt werden. Es kann eine unbegrenzte Anzahl von Reporten generiert werden.

- Eine Einarbeitung bzw. Schulung des Anwenders ist nicht nötig:

  - Die Softwarekomponente für eine Dokumentengenerierung kann vollständig in ein bereits bekanntes System integriert werden.

- Die JasperReports Library ermöglicht die Internationalisierung:

  - Es werden Unicode (UTF-8), ISO-8859-1 (Latin 1) und weitere Encodings unterstützt. Ein Dokument kann so in verschiedenen Sprachen angezeigt werden. Dieses Funktionalität ermöglicht einem Unternehmen große Flexibilität bei Kommunikation mit internationalen Kooperationspartnern und Kunden.

5.4 Vorteile für den Entwickler

Das Framework bietet eine hohe Flexibilität bei der Umsetzung komplexer Dokumentenvorlagen und ermöglicht auch zukünftige Anpassungen an sich ändernden Bedarf:

- Trennung von Layout/Design und Quellcode: Die Dokumentvorlagen können schneller an neue Anforderungen angepasst werden, ohne dass der Entwickler den Quellcode anpassen muss. Beispiele hierfür sind Logos, Text-Passagen oder die Dokumentenorientierung (Portrait, Landscape). Die Änderungen werden nur in der jeweiligen Dokumentvorlage vorgenommen.

- Die Report-Ereignisse können während der Generierung beispielsweise durch:

  - „JasperReports Scriptlets"[17]: Scriplets sind Java-Klassen, die in den Dokumentvorlagen (Master- bzw. Subreport-Vorlagen) definiert werden können. Sie werden beim Auftreten bestimmter Ereignisse, während der Generierung des gesamten Reports, aufgerufen.

  - Logische-Ausdrücke („Java Expression"): Ein Beispiel hierfür ist die Umsetzung bedingter Ausgaben von Report-Elementen (z. B. Felder, Charts oder Subreports). In einem „JasperReports Template" (JRXML) werden die Report-Elemente in s. g. Bands organisiert. Somit kann die Ausgabe der Inhalt eines Bands gesteuert werden.

- Reportdaten können von verschiedenen Quellen bezogen werden (z. B. Datenbank, XML, benutzerdefinierte Quelle):

  - Die Datenquellen werden als Implementierung des Interfaces JRDataSource[18] oder als java.sql.Connection übergeben.

17
Konzept Scriptlets: [2], CHAPTER 8, S.194-199. [4], S.229-232. [5], CHAPTER 17, S.261-274. [6], CHAPTER 11, S.197-201

18
JRDataSource -http://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/JRDataSource.html, (Abruf: 19.05.2012)

6 Umsetzung des Webservices

Die Auswahl einer Softwarearchitektur hängt sehr stark von der wirtschaftlichen Aspekten des Umfeldes und der Anwendungsdomäne, in denen das Softwareprojekt eingesetzt wird, ab. Somit ist die Entscheidung über einen Softwaredesign einer der kritischsten und wichtigsten Punkte im Entwicklungsprozess einer Software. Denn eine bereits festgelegte Softwarearchitektur ist nur mit sehr hohem Aufwand umstellbar.

Das entwickelte Dokumentengenerierungssystem dient als Erweiterung einer bereits vorhanden Datenbankapplikation. Es wird als SOAP basierter Webservice auf einem Applikationsserver (z. B. Apache Tomcat Server) bereitgestellt.

6.1 Architektur

Der Grund der Entscheidung für diese Softwarelösung ist, die Kommunikation zwischen zwei Computernzu ermöglichen, insbesondere wenn beide Systeme auf unterschiedlichen Technologien und Plattformen basieren (Rechner-Rechner-Kommunikation). Dabei kann die Datenbankanwendung beispielsweise in der Programmiersprache C++ geschrieben werden. Mit anderen Worten, Webservices sind für Computer das, was Webseiten für Menschen sind: Ein Anbieter stellt bestimmte Funktionen zur Verfügung, die von einem oder mehreren Nutzern in Anspruch genommen werden können. Die Kommunikation zwischen dem Anbieter der Services und deren Nutzern erfolgt über Nachrichten, die auf XML basieren und zumeist mittels HTTP übertragen werden.

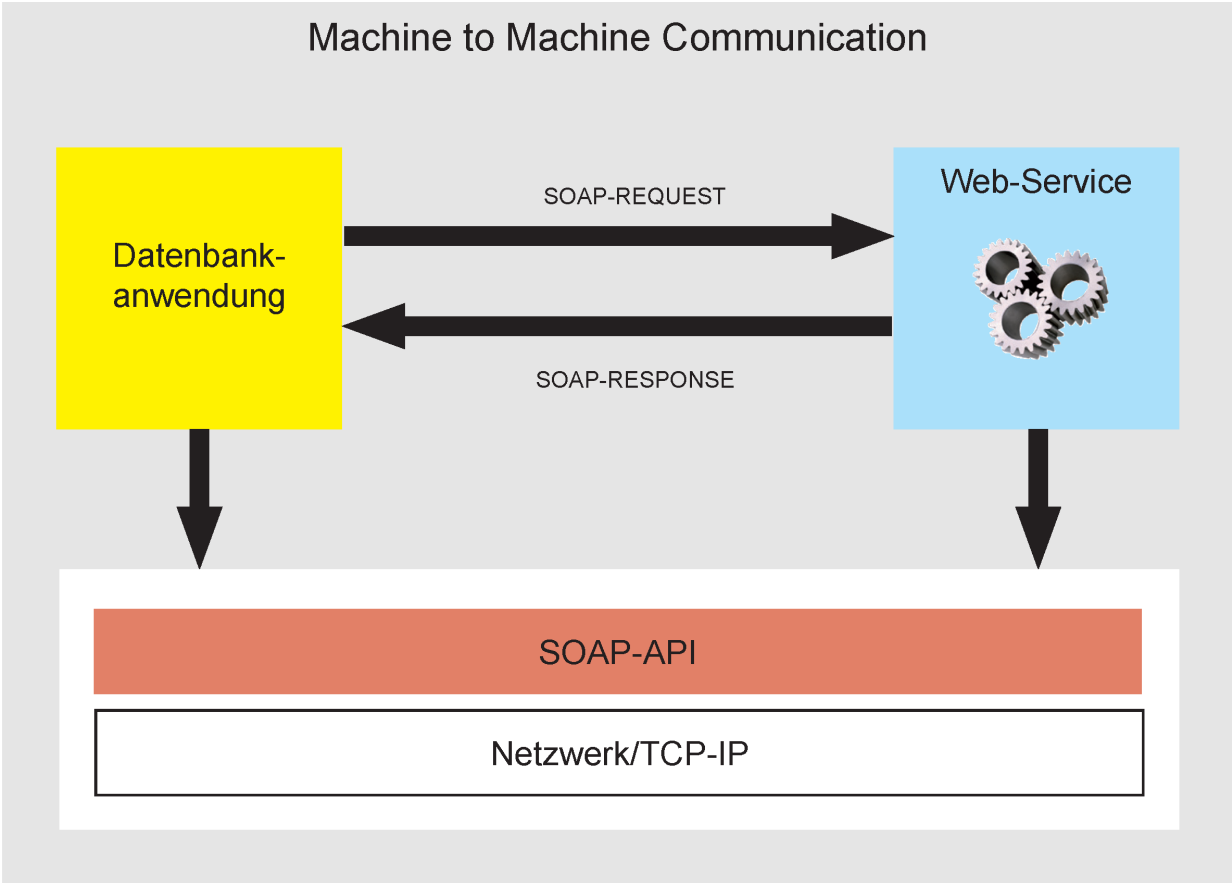

Abbildung 1: Softwarearchitektur

6.2 Aufbau des Softwareprojektes

Das Konzept Modularisierung kommt bei der Entwicklung von komplexen Softwaresystemen vielfach zum Einsatz. Dabei ist es das Ziel, Funktionseinheiten durch eigenständige Softwaremodule zu realisieren. Unter einem Softwaremodul ist eine in sich abgeschlossene Programmeinheit zu verstehen, die bei der Problemlösung eingesetzt werden kann.

Das Stichwort hierfür ist die Objektorientierte Modularisierung. Die Grundidee bei diesem Konzept ist es, solche Softwaremodule als Klassen zu implementieren. Die entwickelte Softwarelösung für die Dokumentengenerierung wurde deshalb folgendermaßen aufgebaut:

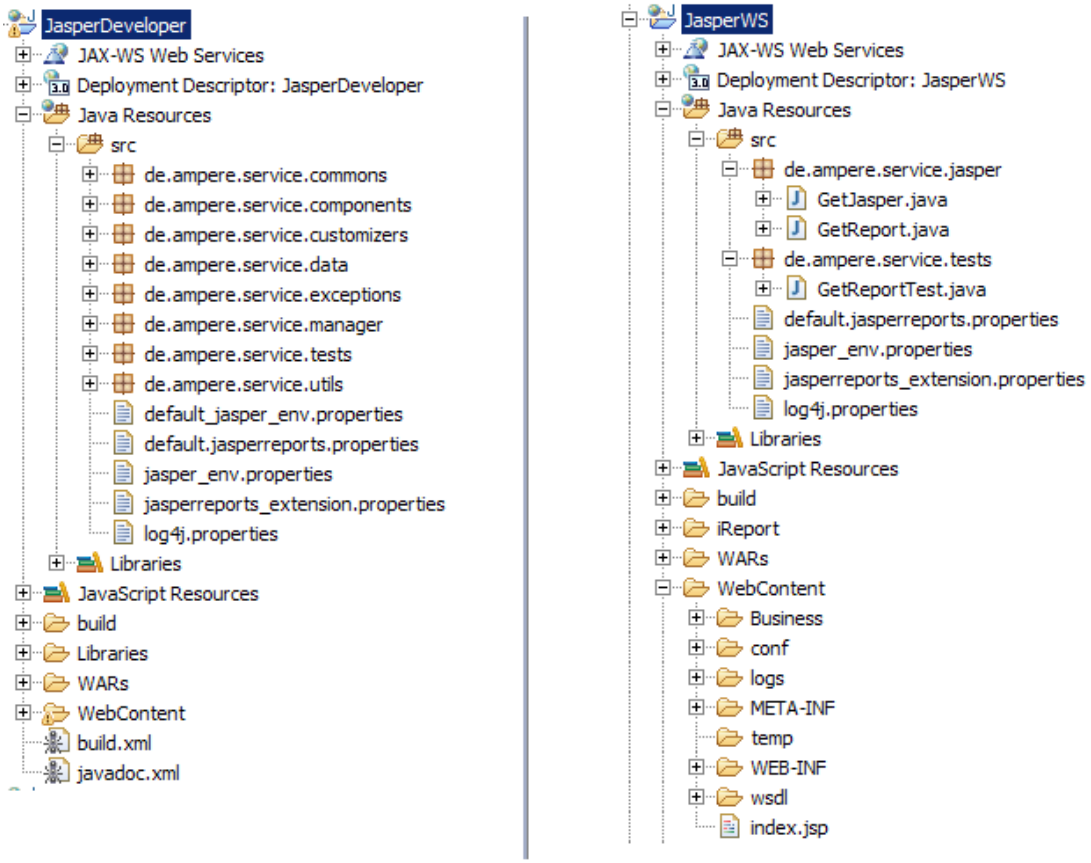

Abbildung 2: Aufbau - Softwareprojekt

6.2.1 JasperDeveloper-Projekt

Das Ziel dieses Projektes ist es, die Webservice-Schnittstelle für die Dokumentengenerierung von der Implementierung der Software-Module zu trennen. Der Vorteil dieser Trennung ist es, dass die Implementierung der Softwaremodulen verbessert oder sogar geändert werden kann, ohne die Webservice-Schnittstelle ändern zu müssen (Information Hiding). Ein weiterer Vorteil dieser Trennung ist, dass die Funktionalitäten der implementierten Softwaremodule leichter zu testen sind.

Um das Geheimnisprinzip auszunutzen, wurden deshalb in diesem Projekt die Java-Packages definiert sowie die Softwaremodulen und Schnittstellen, die für die Entwicklung des Webservices relevant sind, konzipiert und implementiert. Dieser Schutzmechanismus (Datenkapselung) trägt dazu bei, die Benutzung bzw. Verwendung von Softwaremodulen sicherer zu gestalten. Eine solche arbeitsteilige Vorgehensweise stellt sicher, dass der spätere Nutzer das Verhalten der Softwaremodule (Klassen) nicht versehentlich ändern kann.

Nach der Implementierung der Softwaremodule kann der gesamte Source-Code des Projektes als Java Archiv (kurz JAR) exportiert werden. Mit der „Eclipse JAR Exporter-Funktion" lässt sich der gesamte Source-Code eines Eclipse-Projektes einfach und schnell als Java Archiv exportieren. Das Java Archiv kann anschließend als Bibliothek für die Umsetzung des Dokumentengenerierungsprozesses des Webservices genutzt werden. Der nächste Abschnitt stellt das Projekt für die Entwicklung des Webservices vor.

6.2.2 JasperWS-Projekt

Das Ziel dieses Projektes ist es, ein „Java Enterprise Edition Webservice-Modul" mit der JAX-WS (Java API for Web Services) Technologie zu entwickeln. Dabei werden in diesem Projekt eine Ordnerstruktur für die Bereitstellung des Webservices auf einem Applikationsserver aufgebaut, eine Webservice-Schnittstelle definiert sowie die Implementierung dieser Schnittstelle umgesetzt.

Als Grundlage wird die erstellte Klassenbibliothek aus dem vorigen Projekt (JasperDeveloper-Projekt) für die Implementierung der Webservice-Schnittstelle genutzt (Datenkapselung). Des Weiteren werden in diesem Projekt die WSDL (Web Services Description Language) und die WSDD (Web Services Deployment Descriptor) des Webservices erstellt. Dabei kann beispielsweise der „Bottom-Up" bzw. „Top-Down"-Ansatz angewendet werden.

Bei der Erstellung der WSDL und des WSDD des entwickelten Webservices wurde der „Bottom-Up"-Ansatz angewendet.

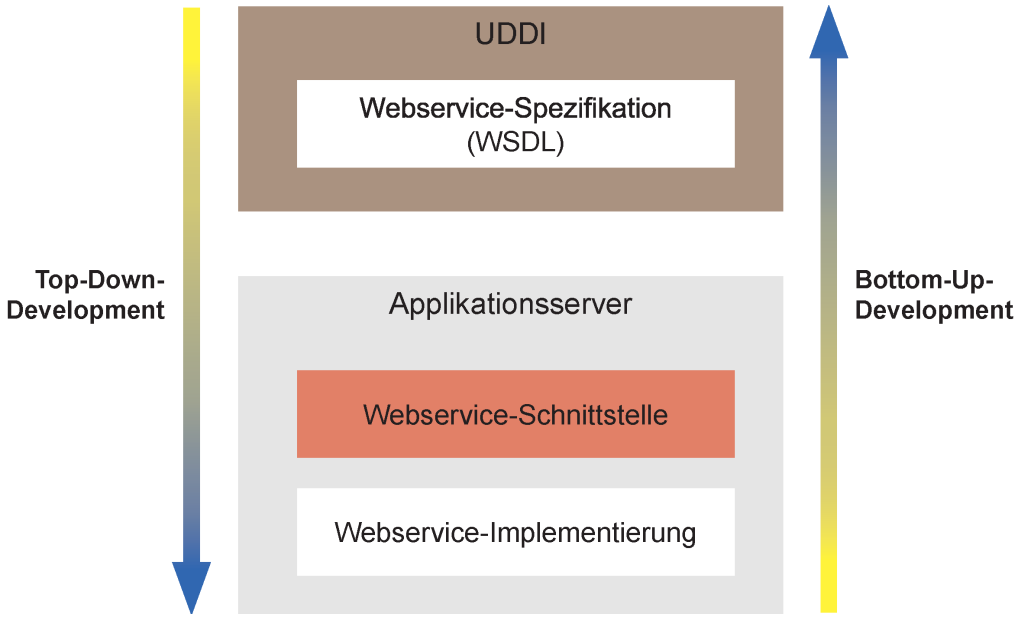

Abbildung 3: BottonUp und TopDown

Nach der Entwicklung des Webservices kann das gesamte Projekt als ein Webarchiv (WAR) exportiert werden. Mit der „Eclipse EE WAR Exporter-Funktion" lässt sich ein EclipseEE-Projekt einfach als Webarchiv exportieren. Das erstellte „Web Archiv" kann über die Oberfläche eines „Application Managers" bzw. über einen FTP-Client auf einem Applikationsserver bereitgestellt werden (Deployment). Nach der Bereitstellung befindet sich der gesamte Inhalt des Webservices in dem entpackten Web Archiv.

6.3 Webservice - Ordnerstruktur

In diesem Abschnitt wird die Ordnerstruktur des Webservices nachdem das Web Archiv (WAR) erfolgreich auf einem Applikationsserver bereitgestellt wurde, ausführlich erklärt.

Der Webserviceordner ist folgendermaßen aufgebaut:

- **Business:** Im „Business"-Ordner befinden sich ein Ordner für die Bilder, ein Ordner für die Styles-Vorlagen, ein Ordner für die digitalen Unterschriften sowie ein Ordner Templates für die umgesetzte Dokumentenvorlagen. Der Ordner Templates enthält einen Ordner Common für die Subreport-Templates (JRXML), die in verschiedene Reportvorlagen verwenden werden können, ein Ordner für die Mastervorlagen sowie die Ordner Kundenreport und Zählervertrag für die umgesetzte Kundenreportvorlage bzw. Vertragsscheiben.

- **conf:** In dem Ordner „conf" befinden sich die Konfigurationsdateien des Webservices bzgl. dem Generierungsprozess.

- **logs:** In dem Ordner „logs" werden die Dokumentengenerierungsprozesse des Webservices geloggt.

- **temp:** Im dem Ordner „temp" werden die serialisierte Objekt vorläufig gespeichert. Diese können beispielsweise über ein Netzwerk übertragt werden.

- **Wsdl:** In dem Ordner „wsdl" befindet sich die Spezifikation der Webservice-Schnittstelle (WSDL).

- **WEB-INF:** In dem Ordner „WEB-INF" befinden sich eine „web.xml" Datei für die Konfiguration eines „Java EE Webservice"-Modul, eine „server-config.wsdd" Datei für die Bereitstellung des Webservices auf einem Applikationsserver, ein Ordner classes für die kompilierte Webservice-Schnittstelle und die implementierte Klasse sowie die JasperReports-Library und die Modul-Bibliothek des Webservices.

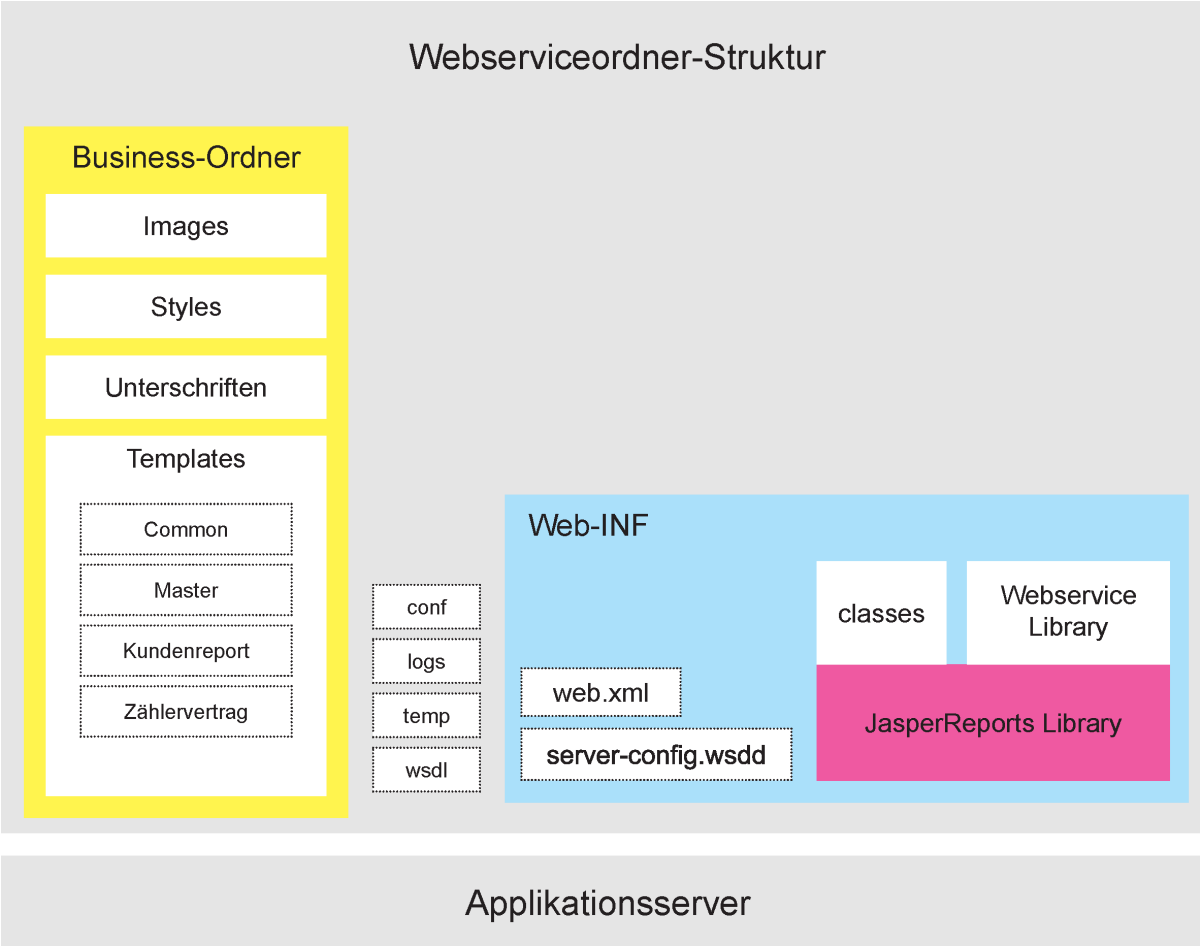

Abbildung 4: Webserviceordner-Struktur

6.4 Webservice – Klassenübersicht

In diesem Abschnitt werden anhand eines Klassendiagramms die Klassen des Webservices vorgestellt. Abbildung 5 auf er folgenden Seite zeigt, wie der Zugriff auf diese Klassen erfolgt.

In dem unten dargestellten Diagramm werden die Klassen von oben nach unten gelesen, auf zwei Ebenen verteilt: In der ersten Ebene werden die Zugriffsklassen organisiert. Das sind Klassen mit deren Hilfe auf Komponenten des Webservices zugriffen wird. Die zweite Ebene enthält die Klassen, mit deren Hilfe die Logik des Dokumentenerzeugungsprozesses umgesetzt wird.

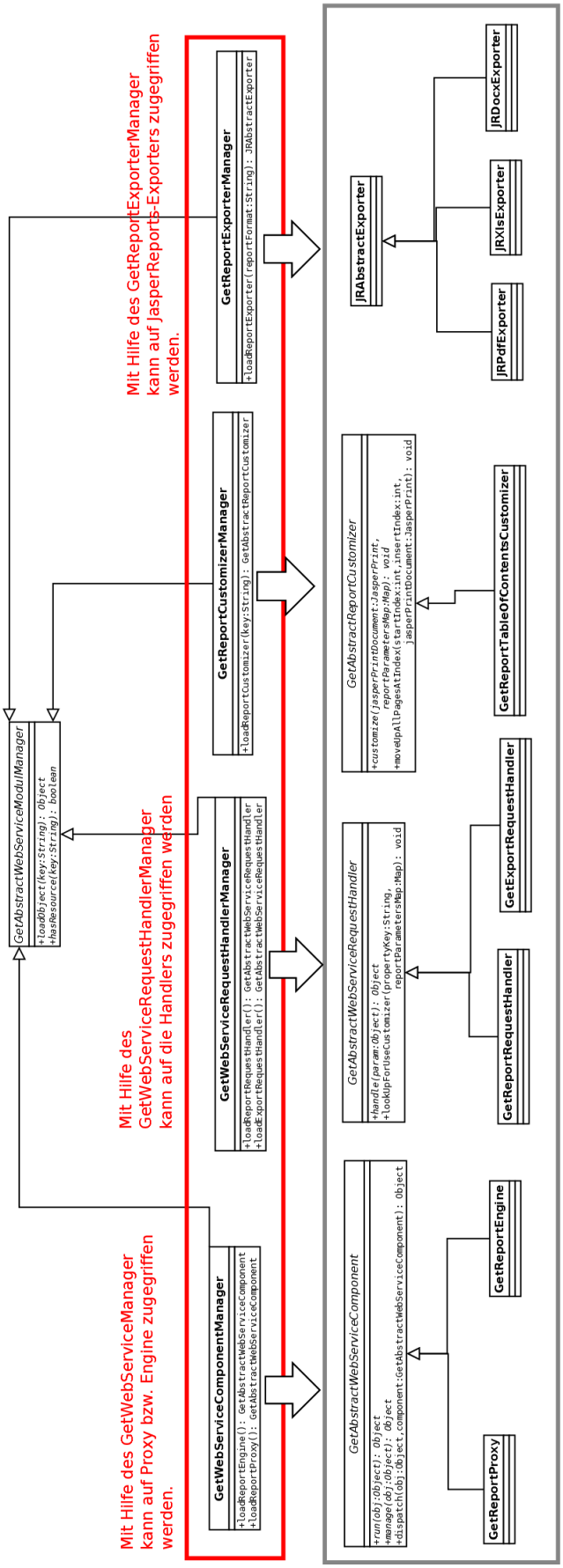

Abbildung 5: Webservice - Klassenübersicht

6.5 JasperReports API Overview

Bei der Dokumentenerzeugung wird in der Regel wird von einer JRXML Datei ausgegangen. Die JRXML Datei wird mit Hilfe der Klasse net.sf.jasperreports.engine.xml.JRXMLLoader in ein Objekt der Klasse net.sf.jasperreports.engine.design.JasperDesign umgewandelt, welches mit Hilfe der Klasse net.sf.jasperreports.engine.JasperCompileManager in ein Objekt der Klasse net.sf.jasperreports.engine.JasperReport compiliert wird. Das compilierte Objekt kann wiederum mit Hilfe der Klasse JasperFillManager gefüllt werden. Das resultierende JasperPrint-Objekt kann anschließend exportiert bzw. angezeigt werden (siehe Abbildung 6).

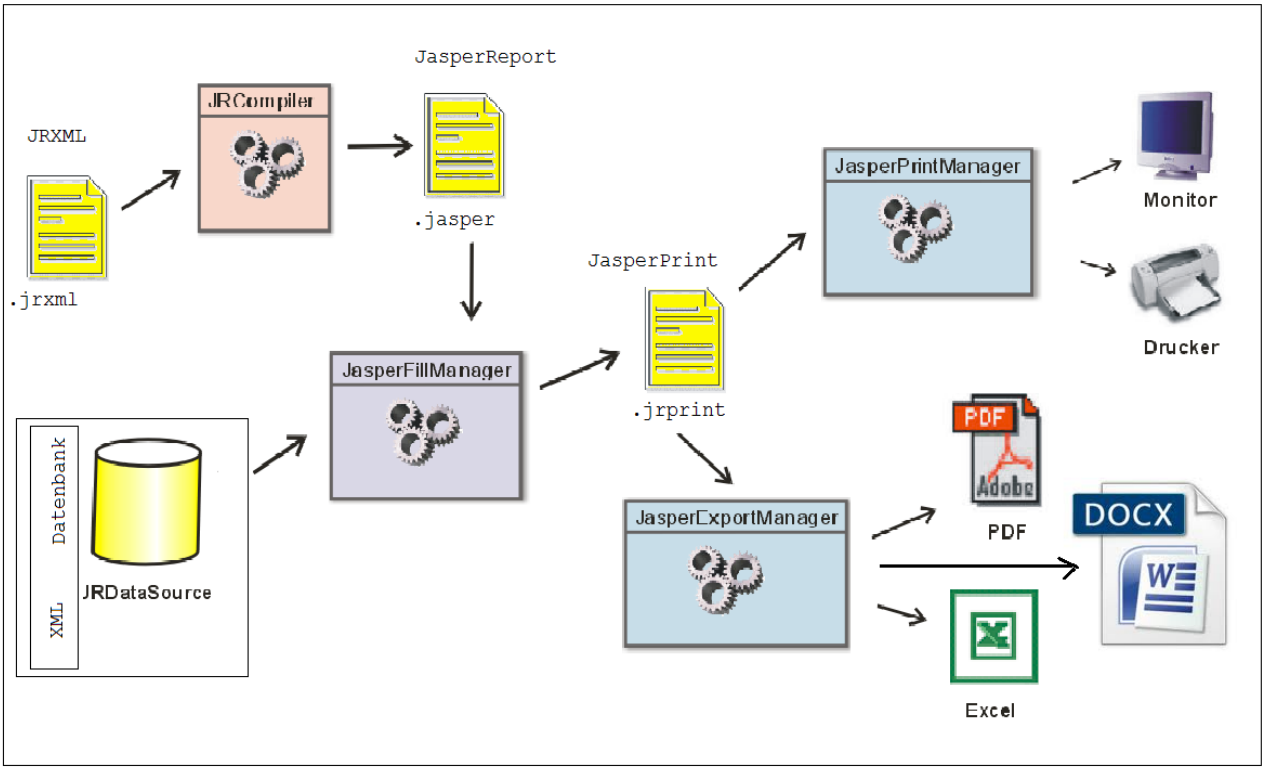

Abbildung 6: Ablauf Dokumentenerzeugung

6.6 Komponenten des Webservices

In diesem Abschnitt werden die Softwarekomponenten des entwickelten Dokumentengenerierungssystems vorgestellt.

Die entwickelte Softwarelösung für die Dokumentengenerierung geht hauptsächlich von zwei Softwarekomponenten aus: eine Klasse GetReportProxy und eine Klasse GetReportEngine. Die beide Softwarekomponenten werden als Subklasse der Superklasse GetAbstractWebServiceComponent implementiert.

Die Idee bei dieser Vorgehensweise ist es, das Geheimnisprinzip und das Konzept Datenkapselung bei einer objektorientierten Softwareentwicklung auszunutzen. Dabei sollen Details über den inneren Aufbau vor dem Aufrufer verborgt werden. Der Zugriff auf Methoden erfolgt nur über die vorgesehene Schnittstelle. Das hat den Vorteil, dass ein Objekt als Software-Baustein genau das Verhalten hat, das von den Entwicklern des Bausteins realisiert wurde.

- **Funktionsweise des Webservices**

Die Softwarekomponente GetReportProxy ist zuständig für die Verarbeitung einer Reportanfrage. Der GetReportProxy schickt eine Exportanfrage an die GetReportEngine. Die Softwarekomponente GetReportEngine ist zuständig für die Verarbeitung einer Exportanfrage. Die GetReportEngine kann dabei ein Dokument in unterschiedlichen Formate exportieren (siehe Abbildung 7) .

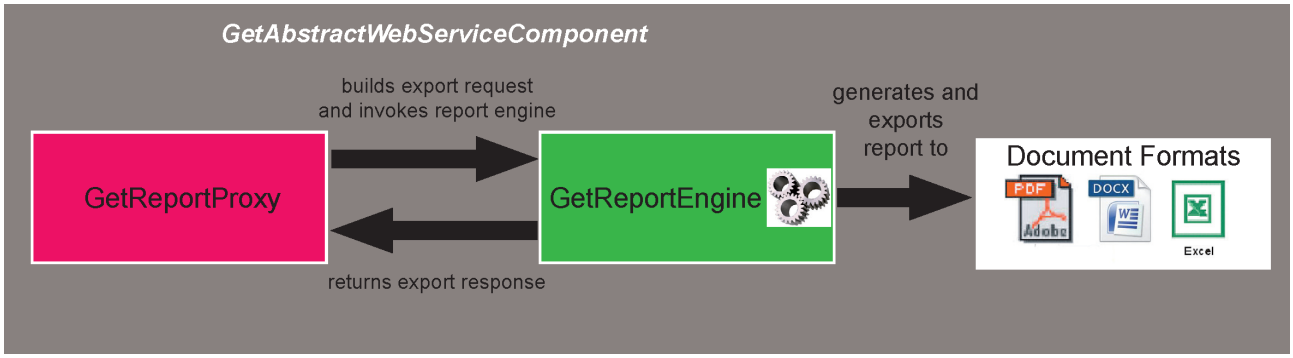

Abbildung 7: GetReportProxy und GetReportEngine

6.7 Aufruf des Webservices

Speziell bei der hier entwickelten Lösung bietet eine Datenbankapplikation dem Anwender über ein Dialogfenster die Möglichkeit, alle Request-Parameter zu definieren. Die Datenbankapplikation nimmt die vom Anwender definierten Parameter entgegen und übermittelt diese, inklusive der im Dokument auszugebenden Daten, an den Webservice. Die Daten werden in einem SOAP-Request als XML-Dokument übergeben. Nach einer erfolgreichen Erzeugung des Dokumenten schickt der Webservice ein SOAP Response zurück (siehe Abbildung 8).

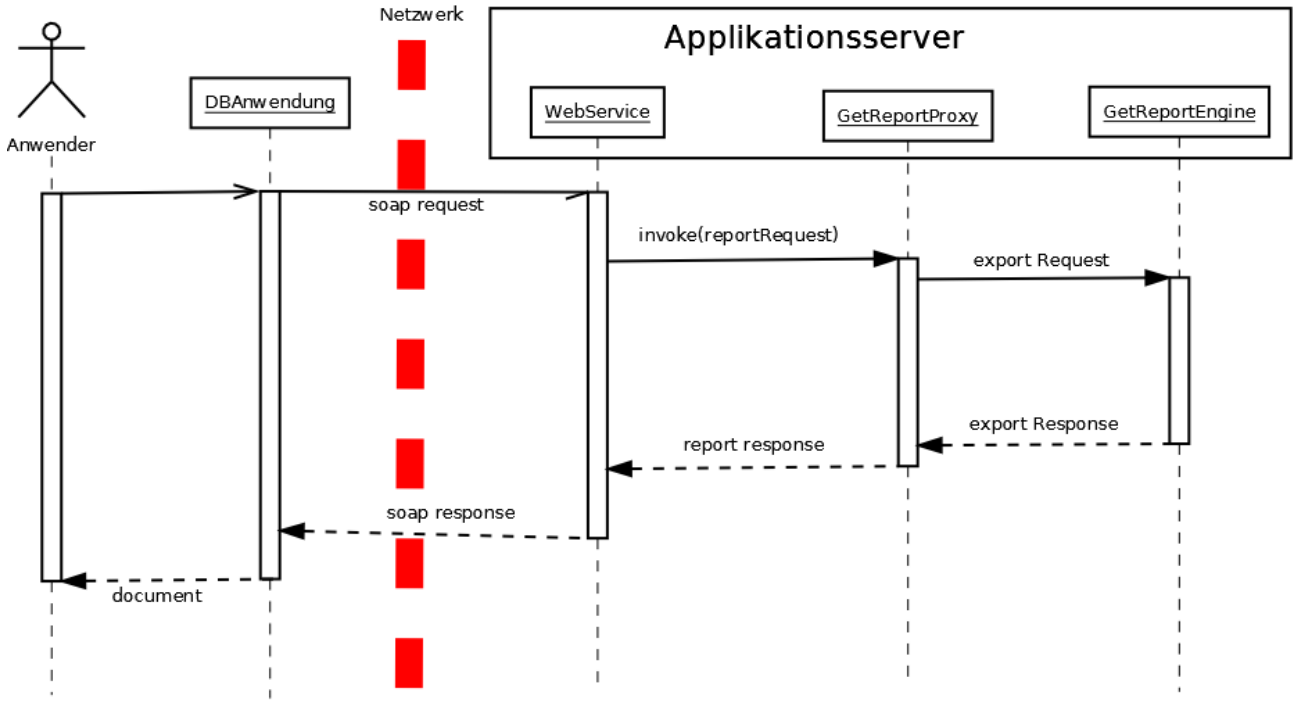

Abbildung 8: Sequenzdiagramm

- **Funktionsweise des GetReportProxys**

Die Hauptaufgabe der Komponente GetReportProxy besteht darin, die Anfrage an den Webservice zu verarbeiten. Dabei wird die angeforderte Vorlage als JasperPrint Dokument erzeugt und bei Bedarf angepasst, bevor das Dokument an den Exporter geschickt wird (siehe Abbildung 9).

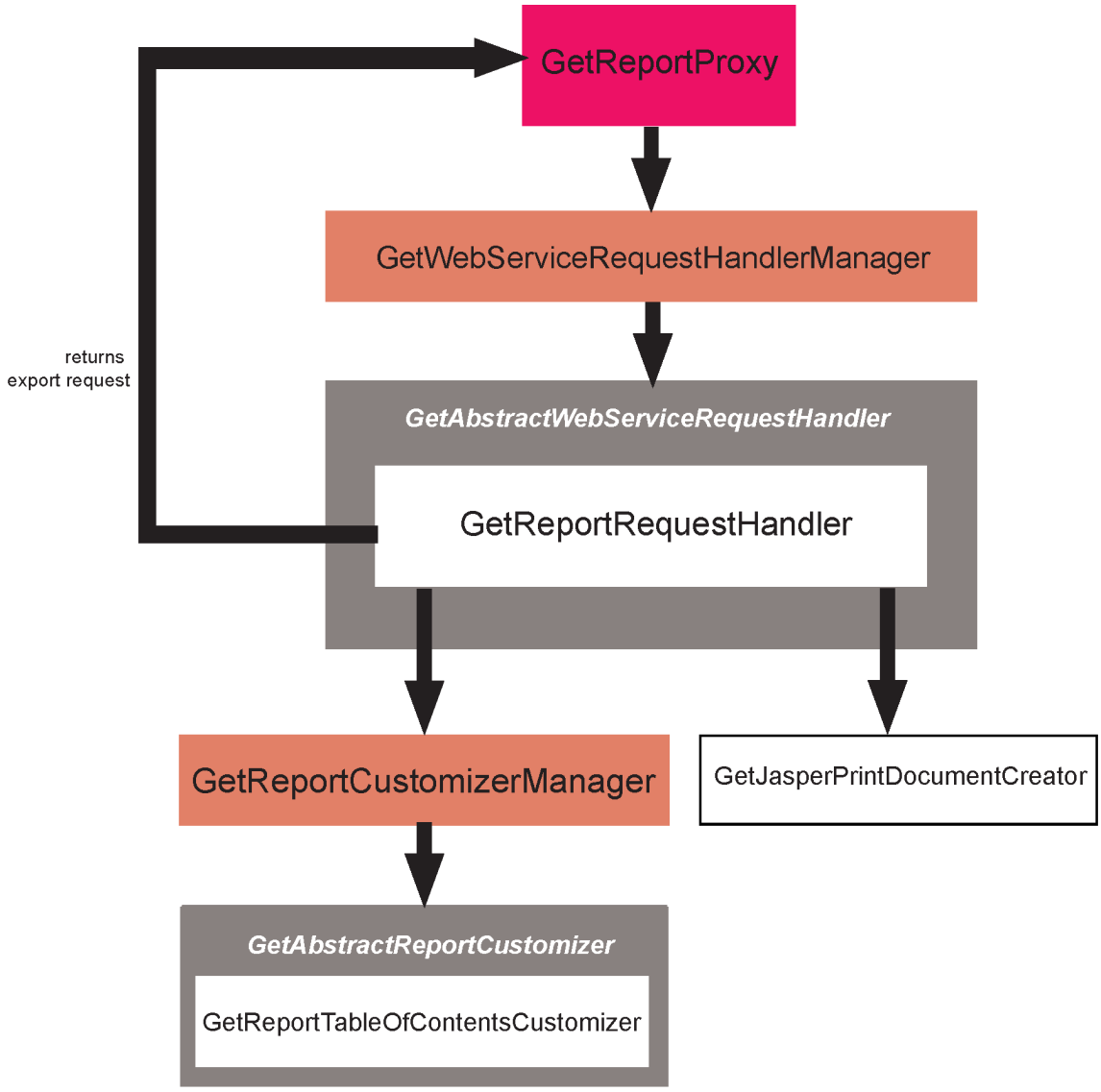

Abbildung 9: GetReportProxy

- **Funktionsweise der GetReportEngine**

Die Komponente GetReportEngine hat die Aufgabe ein bereits erzeugtes Dokument in den gewünschten Format zu exportieren. Dabei wird zuerst überprüft ob der Webservice einen entsprechenden JasperReportExporter unterstützt (siehe Abbildung 10).

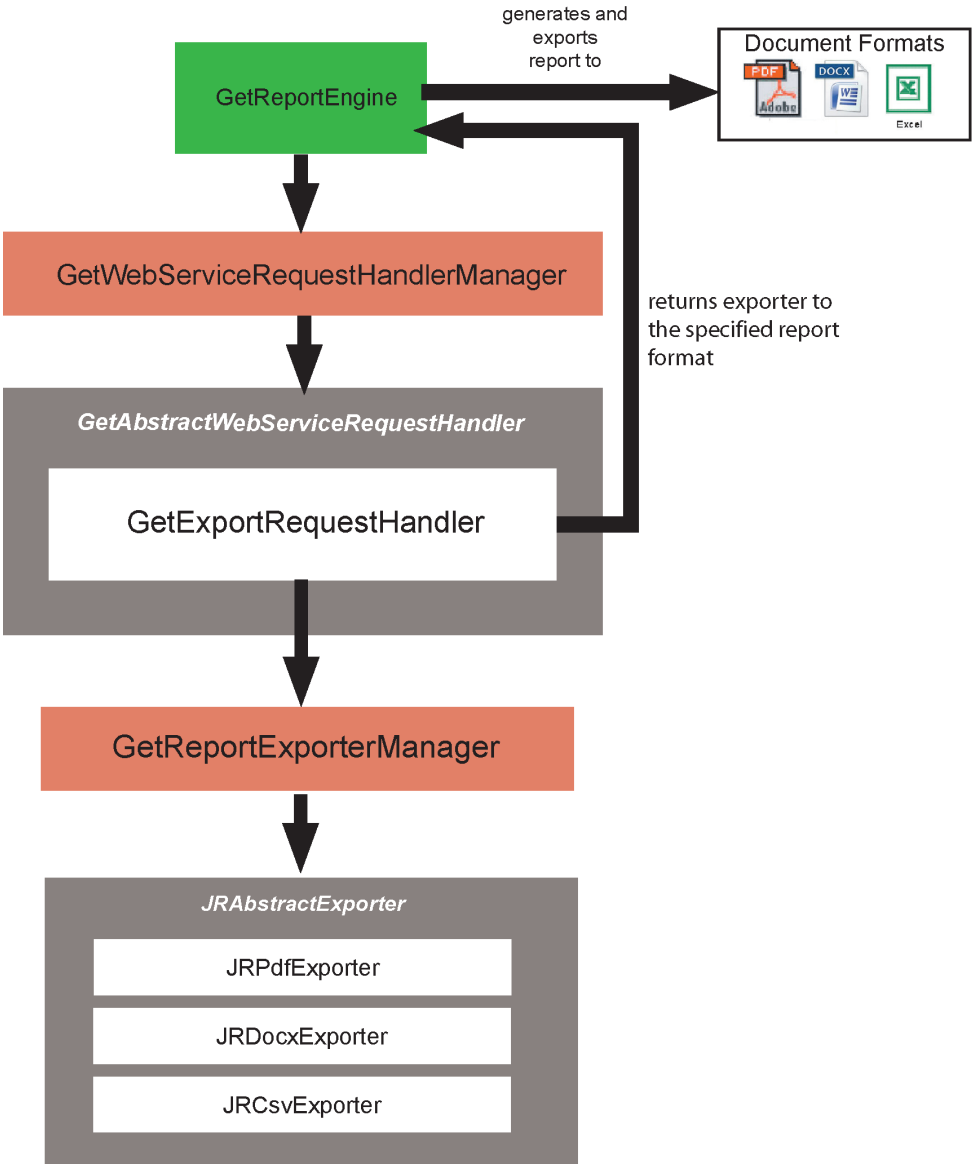

Abbildung 10: GetReportEngine

6.8 Aktivitätsdiagramm

Das in Abbildung 11 gezeigte Aktivitätsdiagramm beschreibt die einzelnen Aktivitäten des Webservices von der Entgegennahme der Anfrage bis zur Antwort auf die Anfrage.

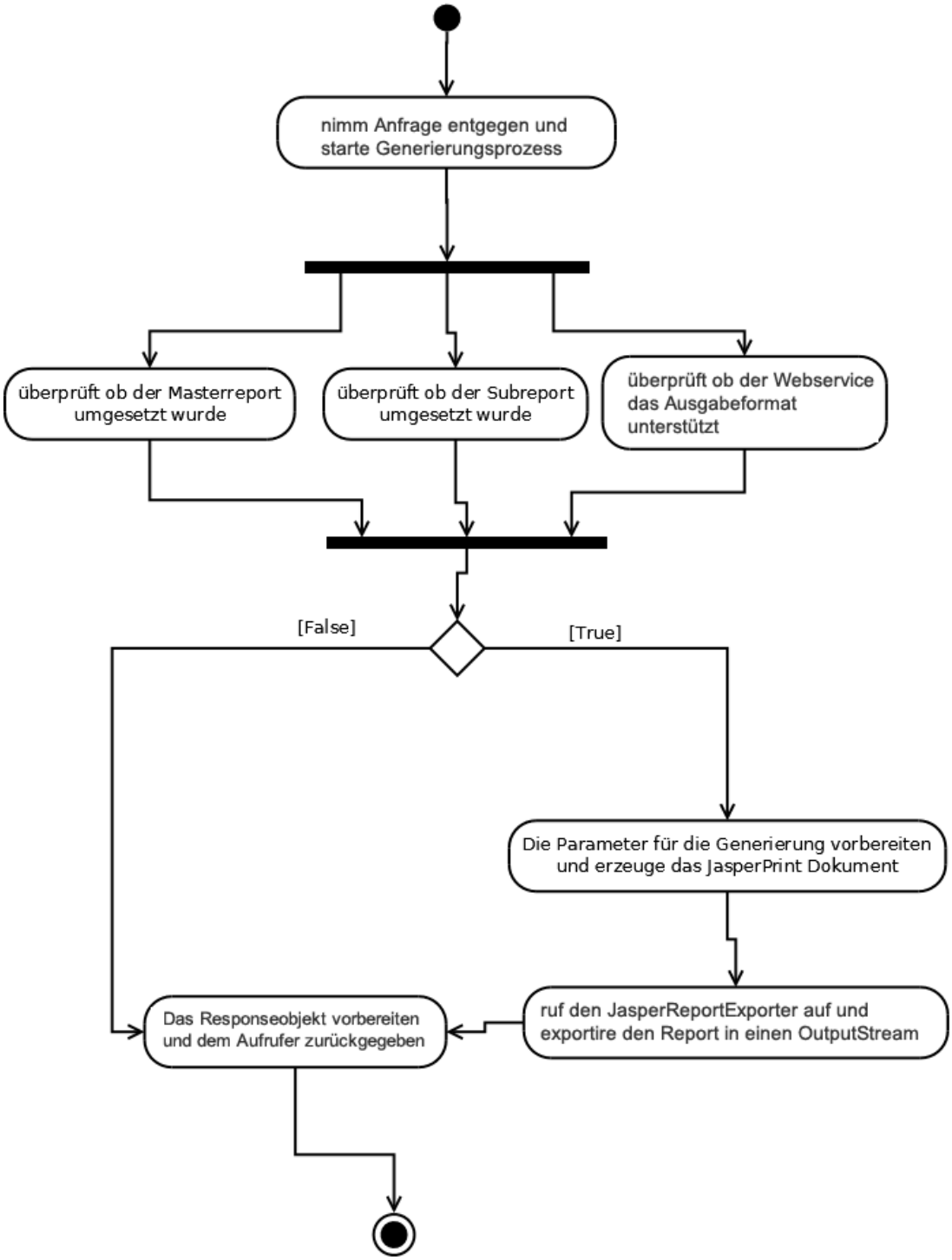

Abbildung 11: Webservice - Aktivitätsdiagramm

7 Umsetzung der Dokumentenvorlagen

In diesem Kapitel werden die Dokumentenvorlagen, die im Rahmen dieser Arbeit mit dem Webservice umgesetzt wurden, vorgestellt. Zunächst werden die Begriffe und Abkürzungen, die im diesem Kapitel verwendet werden, erklärt.

7.1 Begriffe und Abkürzungen

- **Musterdesign**
  - definiert Prinzipien bzw. Regeln, die für die Erstellung einer JasperReports-Vorlage zu beachten sind.
- **SVK**
  - Sondervertragskunden
- **GTK**
  - Gewerbetarifkunden
- **GTK Online**
  - Gewerbetarifkunden mit Onlineverträgen

7.2 Ausgangssituation

Die Dokumentenvorlagen, die von der Ampere AG, im Rahmen ihrer Dienstleistungen eingesetzt werden, sind statische MS Word-Vorlagen, die mit Feldfunktionen hinterlegt sind. Das bedeutet, dass für jede Kundenart unterschiedliche Word-Vorlagen erstellt werden. Dabei stellt jede Word-Vorlage ein Master dar.

7.3 Musterdesign

Die Erarbeitung eines Lösungsdesigns (Muster-Design), welches die Grundlage für die Umsetzung der Dokumentenvorlagen des Unternehmens bildete, spielte eine große Rolle. Das Design nutzt das Konzept Master-/Subreport vom JasperReports Framework – d. h. dass das Report-Layout (Master) wird von der Organisation der Informationen für das fertige Dokument (Subreport) getrennt. Ein weiterer Aspekt dieses Konzeptes ist die Wiederverwendbarkeit von Subreports. Je nach Report z. B. (Vertragsschreiben, Kundenreport) bzw. Kundenart z. B. (GTK, SVK oder Online-GTK) kann ein Subreport wiederverwendet werden (siehe Abbildung 12).

Abbildung 12: Musterdesign

7.4 Vertragsschreiben

Bei dem alten Vertragsscheiben wurden für jede Kundenart unterschiedliche Word-Vorlage mit eigenen Kopfzeile, Briefkopf und Fußzeile erstellt. Diese Lösung enthält redundante Informationen. Egal ob es sich um einen GTK-, SVK oder GTK Online-Kunde handelte, die Informationen in den Fuß-/Kopfzeilen blieben immer gleich. Der Briefkopf wurde mit Feldfunktionen gefüllt (Makroprogrammierung). Um diese Redundanz und vor allem das Erstellen von mehreren Master zu vermeiden sollen nun die alten Vorlagen von einer einzige und flexibleren Lösung ersetzt werden.

Mit dem „JasperReports Framework" wurde ein „Master Template" (JRXML Datei) mit Design und Layout für verschiedene Kundenarten (SVK, GTK und GTK Online) erstellt. Um das Konzept Master-/Subreport auszunutzen, wurde das Generieren des Briefkopfes in eigenem „Subreport Template" (ebenfalls eine JRXML Datei) ausgelagert. So werden die Daten, je nach Kundenart, zum Zeitpunkt der Generierung des gesamten Reports in ein java.util.Map gespeichert und per Parameterübergabe an den Subreport (Briefkopf) weitergeleitet (siehe Abbildung 13).

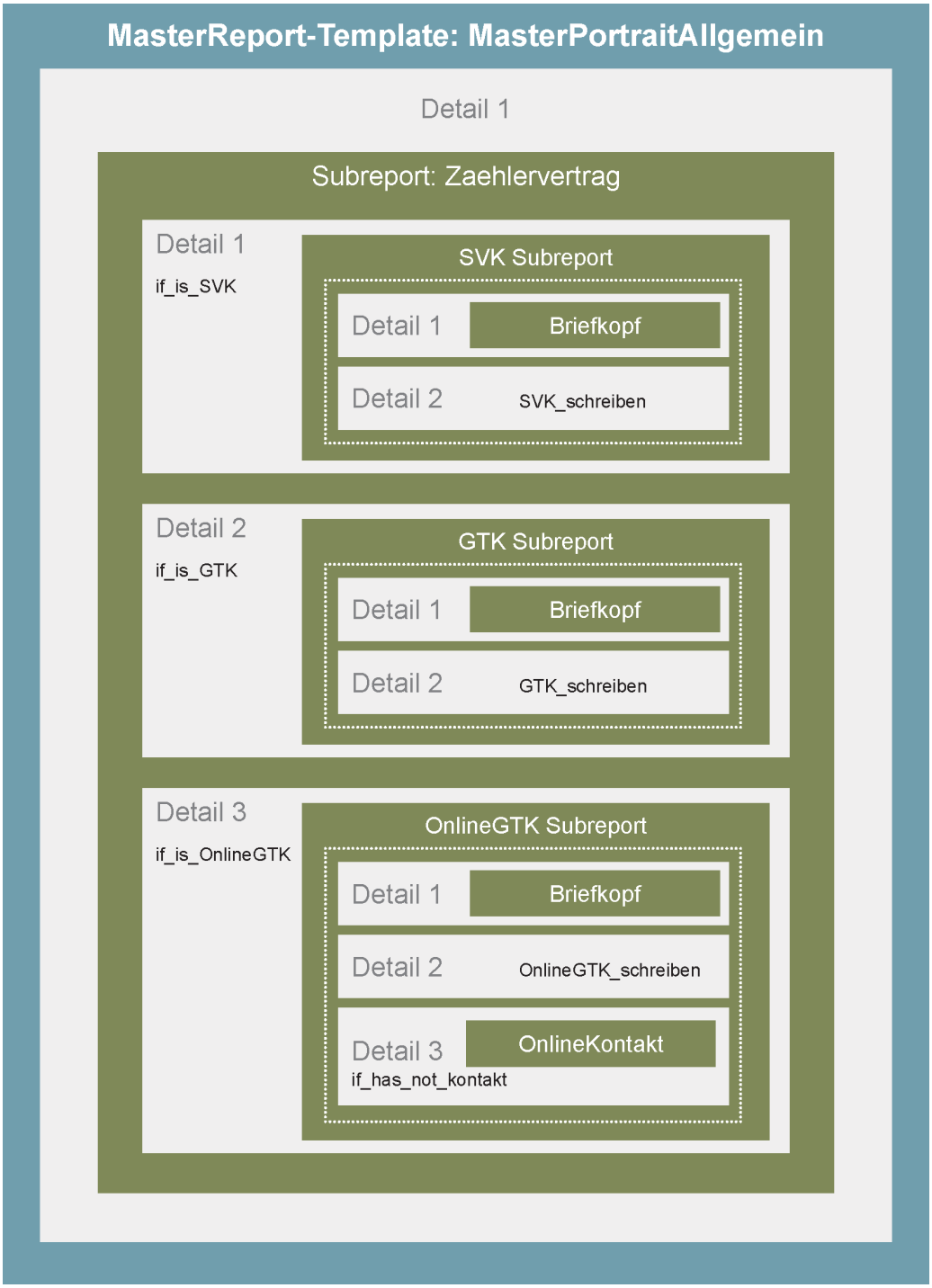

Abbildung 13: Mustervorlage für Vertragsschreiben

7.5 Kundenreport

Der Kundenreport ist einer der wichtigsten Berichte, den die Ampere AG ihren Kunden als Dienstleistung zur Verfügung stellt. Es handelt es sich bei diesem Bericht um komplexe Auswertungen (Analysen) der Abnahmestellen (Standorte) eines Kunden.

Das Ziel der Umsetzung dieser Vorlage ist es, den Kunden einen Überblick über seine Abnahmestellen zu liefern. Dabei kann es sich z. B. um den Verbrauch, Strom-/Gaskosten oder die laufenden Energielieferverträge seiner Standorte handeln. Die Daten, welchen von der Ampere AG im Rahmen ihrer Dienstleistungen gepflegt und verwaltet werden, bilden die Grundlage für das Generieren eines Kundenreports. Im nächsten Abschnitt wird der Aufbau eines Kundenreports vorgestellt.

Der Kundenreport stellt, neben den Allgemeinen Daten, wie Zählernummer, Leistungsmessung und Spannungs-/Druckebene, auch Preise und Kondition von Strom-/Gaslieferverträgen, Verbräuche und Kosten nach Standorten bereit. Angesichts der Anforderungen und Bedürfnisse einzelner Kunden ist der Kundenreport modular wie folgt aufgebaut:

1. Kundendaten
2. Standorte
3. Verbrauch und Kosten nach Standorten
4. Einzelaufstellung Standorte
    - 4.1. Ausgangspreise
    - 4.2. Energielieferverträge
    - 4.3. Entwicklung Verbrauch

Jedes dieser Module ist als eigenständiges „Subreport Template" (JRXML) definiert. Es kann auf Wunsch des Kunden über eine Anwendung für das Generieren eines Kundenreports ausgewählt werden (Abbildung 14).

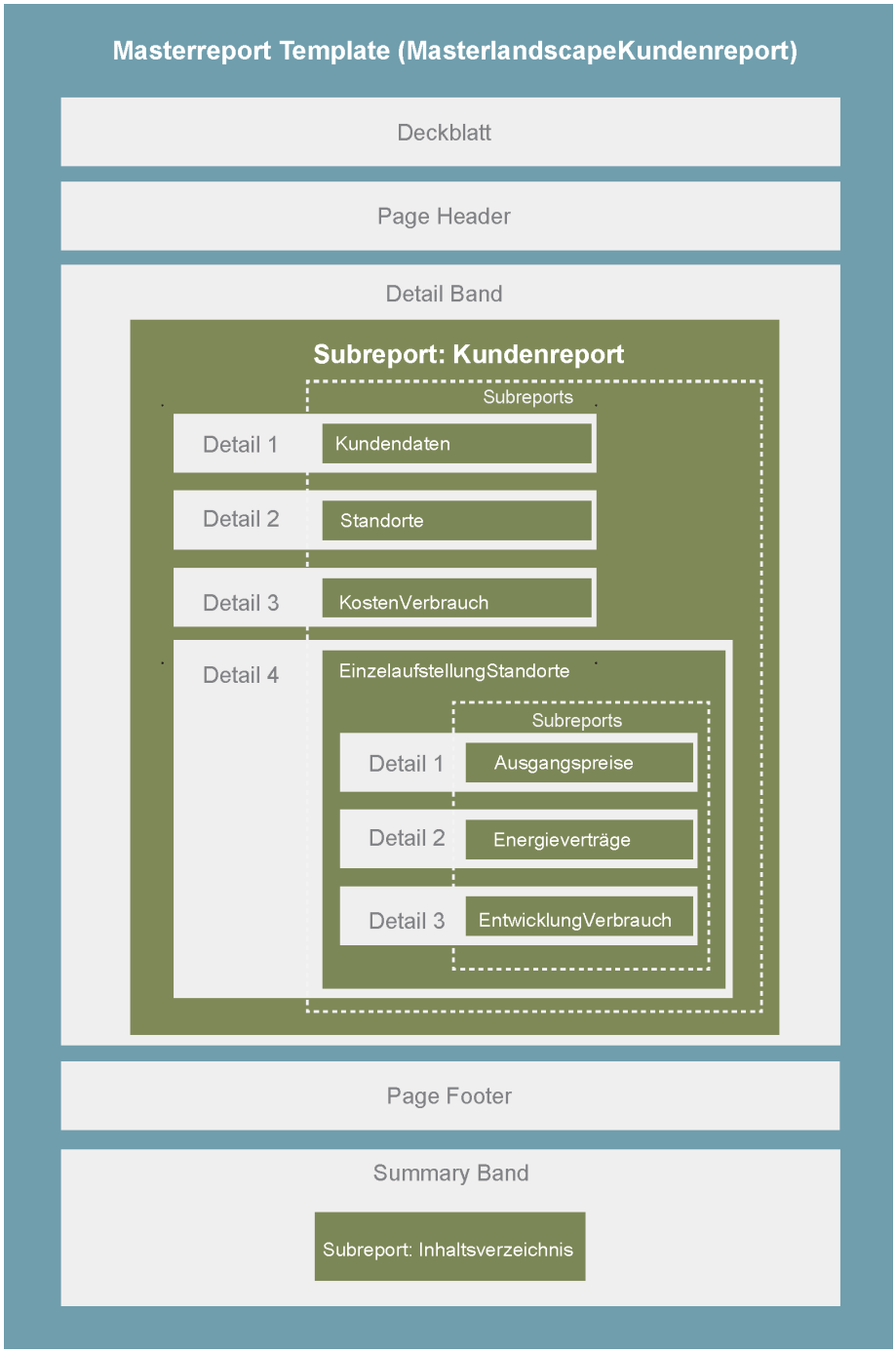

Abbildung 14: Mustervorlage für Kundenreport

8 Fazit

Das Ziel dieser Arbeit, die bisherige aufwändige Arbeitsweise bei der Erstellung von Kundenreporten der Ampere AG durch eine effizientere Methode zu ersetzen, ist erreicht worden.

Die Erzeugung von Kundenreporten durch den Einsatz von JasperReports über eine Webservice-Schnittstelle hat den täglichen Arbeitsaufwand der Ampere-Kundendienstmitarbeiter erheblich reduziert. Wurden bisher für die Erzeugung eines Report mittels MS-Word viele einzelne Arbeitsvorgänge benötigt, ist nun das Dokument, nach Eingabe weniger Parameter, per Knopfdruck innerhalb kürzester Zeit fertig.

Mit der, in dieser Arbeit vorgestellten, Technologie wurden bereits mehrere Dokumentvorlagen erfolgreich umgesetzt. Diese wurden unternehmensintern ausführlich und erfolgreich getestet. Alle wichtigen Vorlagen sollen in Zukunft auf diese Weise realisiert werden.

Die Mitarbeiter des Unternehmens schätzen vor allem , dass sie nicht mehr mit dem Prozess der Dokumentengenerierung konfrontieret sind, sondern nur noch die gewünschten Daten zuliefern müssen.

Die Entscheidung für eine serviceorientierte Softwarearchitektur hat sich in der Praxis als richtig erwiesen.

Die Konzeptentscheidungen:

a) die Erstellung der Dokumentenvorlagen von der Organisation der Datensätze zu trennen und

b) die Dokumentenerzeugung über eine Webserviceschnittstelle zu realisieren

erhöhen die Sicherheit des gesamten Systems.

Das System ist auch für zukünftige Anwendungen nutzbar und ermöglicht durch seine Flexibilität fortlaufende Anpassungen an den steigenden Bedarf.

Die Arbeit der IT-Abteilung wird durch die Implementierung des vorgestellten Systems erleichtert, da neue Vorlagen in Zukunft auf den bereits definierten Mustern basieren können. Durch den modularen Aufbau ist die Änderung einzelner Komponenten des Systems leicht möglich.

Anhang

Anhangverzeichnis

Einblick in JasperReports und iReport-Designer.....46

Abbildungsverzeichnis..... 52

Quelltextverzeichnis..... 59

CD-ROM..... 74

Literaturverzeichnis.....75

Einblick in JasperReports und iReport-Designer

JasperReports gehört zu der „Business Intelligence"-Produktfamilie des Unternehmens Jaspersoft und ist weltweit eine der populärsten „Open Source Reporting Library", die in Java geschrieben wurde. JasperReports wurde im Jahr 2001 von Teodor Danciu[19] auf SourceForge[20] veröffentlicht. Das Framework ermöglicht den Zugriff auf verschiedenen Datenquellen (z. B. XML, Datenbank) und produziert pixelgenaue Dokumente, die in eine Vielzahl von Formaten (z. B. PDF, Excel, Word) exportiert werden können. Mit Hilfe des iReport-Designers lassen sich sehr anspruchsvolle und flexible „JasperReports Templates" (Report Design/Layouts) mit Grafiken, Barcodes etc. erstellen.

JasperReports und iReport-Designerwerden unter der Open-Source Lizenz „GNU General Public License" (GPLv3)[21] entwickelt.

---

19 Teodor Danciu ist Mitgründer des Unternehmens „Jaspersoft", Software-Autor und Architekt von „JasperReports". http://www.jaspersoft.com/management-team#tdanciu (Abruf: 19.05.2012).

20 SourceForge: http://sourceforge.net/ (Abruf: 19.05.2012)

21 GPLv3 http://www.gnu.org/licenses/gpl-3.0.html (Abruf: 19.05.2012). oder http://jasperforge.org/ (Abruf: 19.05.2012).

JasperReports Template

Unter einem JasperReports Template ist im Allgemeinen eine JRXML Datei zu verstehen. Eine JRXML ist eine valide XML-artige Datei mit der Erweiterung .jrxml. In einer JRXML Datei werden die Struktur des Dokuments inklusive aller im Dokument auszugebenden Informationen (z. B. Texte, Barcodes oder Grafiken) definiert.

```
<!ELEMENT jasperReport (property*, import*, reportFont*, style*,
      subDataset*,   parameter*,   queryString?,   field*,   sortField*,
variable*,
      filterExpression?, group*,
      background?,   title?,   pageHeader?,   columnHeader?,   detail?,
columnFooter?,
      pageFooter?, lastPageFooter?, summary?)>
<!ATTLIST jasperReport
      name CDATA #REQUIRED
      language CDATA "java"
      columnCount NMTOKEN "1"
      printOrder (Vertical | Horizontal) "Vertical"
      pageWidth NMTOKEN "595"
      pageHeight NMTOKEN "842"
      orientation (Portrait | Landscape) "Portrait"
      whenNoDataType (NoPages | BlankPage | AllSectionsNoDetail)
      "NoPages"
      columnWidth NMTOKEN "555"
      columnSpacing NMTOKEN "0"leftMargin NMTOKEN "20"
      rightMargin NMTOKEN "20"
      topMargin NMTOKEN "30"bottomMargin NMTOKEN "30"
      isTitleNewPage (true | false) "false"
      isSummaryNewPage (true | false) "false"
      isSummaryWithPageHeaderAndFooter (true | false) "false"
      isFloatColumnFooter (true | false) "false"
      scriptletClass CDATA #IMPLIED
      resourceBundle CDATA #IMPLIED
      whenResourceMissingType  (Null  |  Empty  |JasperReports  -  JRXML
Syntax1 Key | Error) "Null"isIgnorePagination (true | false) "false"
      formatFactoryClass CDATA #IMPLIED>
```

Quelltext 1: JasperReports - JRXML Syntax

```
<!ELEMENT subreport (reportElement, parametersMapExpression?,
subreportParameter*,
      (connectionExpression | dataSourceExpression)?, returnValue*,
subreportExpression?)>
<!ATTLIST subreport
isUsingCache (true | false) #IMPLIED
>
<!ELEMENT parametersMapExpression (#PCDATA)>
<!ELEMENT subreportParameter (subreportParameterExpression?)>
<!ATTLIST subreportParameter
name CDATA #REQUIRED
>
<!ELEMENT subreportParameterExpression (#PCDATA)>
<!ELEMENT returnValue EMPTY>
<!ATTLIST returnValue
subreportVariable CDATA #IMPLIED
toVariable CDATA #IMPLIED
calculation (Nothing | Count | DistinctCount | Sum | Average | Lowest |
Highest
| StandardDeviation | Variance) "Nothing"
incrementerFactoryClass CDATA #IMPLIED
>
<!ELEMENT connectionExpression (#PCDATA)>
<!ELEMENT dataSourceExpression (#PCDATA)>
<!ELEMENT subreportExpression (#PCDATA)>
<!ATTLIST subreportExpression
class (java.lang.String | java.io.File | java.net.URL |
java.io.InputStream
| net.sf.jasperreports.engine.JasperReport |
dori.jasper.engine.JasperReport)
"java.lang.String"
>
```

Quelltext 2: SubReport Element - JRXML Syntax

JasperReports Abhängigkeiten

JRE	• Java Runtime Environment 1.4 or höher
Commons	• Jakarta Commons BeanUtils Component (version 1.7 oder höher) http://jakarta.apache.org/commons/beanutils/ • Jakarta Commons Collections Component (version 2.1 oder höher) http://jakarta.apache.org/commons/collections/ • Jakarta Commons Javaflow (Sandbox version) http://jakarta.apache.org/commons/sandbox/javaflow/ • Jakarta Commons Logging Component (version 1.0 oder höher) http://jakarta.apache.org/commons/logging/

JRXML	• JAXP 1.1 XML Parser
	• Jakarta Commons Digester (version 1.7 oder höher) http://jakarta.apache.org/commons/digester/
	• Für die Report Übersetzung einen der folgenden Compiler:
	  ○ Eclipse JDT Java Compiler (recommended) http://www.eclipse.org/jdt/
	  ○ JDK 1.4 oder höher

JDBC	• JDBC 2.0 Treiber
XLS	• Jakarta POI (version 3.0.1 or later) http://jakarta.apache.org/poi/ • JExcelApi (version 2.6 or later) http://jexcelapi.sourceforge.net/
Charts	• JFreeChart (>= Version 1.0.0) http://www.jfree.org/jfreechart/ JCommon – für JFreeChart http://www.jfree.org/jcommon/ • Batik SVG Toolkit (>= Version 1.7) – erforderlich, wenn Charts als SVG gerendert werden sollen. http://xmlgraphics.apache.org/batik/
PDF	• iText - Freie Java-PDF library (>= version 1.3.1) http://www.lowagie.com/iText/

IReport-Designer Frontend

iReport-Designer ist der „Open Source Report Designer" für JasperReports. Mit iReport-Designer lassen sich sehr anspruchsvolle Report-Vorlagen mit Grafiken, Kreuztabellen etc. erstellen. Der Zugriff auf Report-Datenquellen (z. B. Datenbank, XML) können ebenfalls mit iReport-Designer realisiert werden. Eine der wichtigsten Funktionalitäten des Report-Designers ist die Unterstützung des „JasperReports"-Compilers und des Exporters. Es können „JasperReports Templates" (JRXML) einfach kompiliert und exportiert werden. Mit dem integrierten Viewer können Dokumente in verschiedenen Formaten (z. B. PDF, XLS, CSV, DOCX) angezeigt werden.

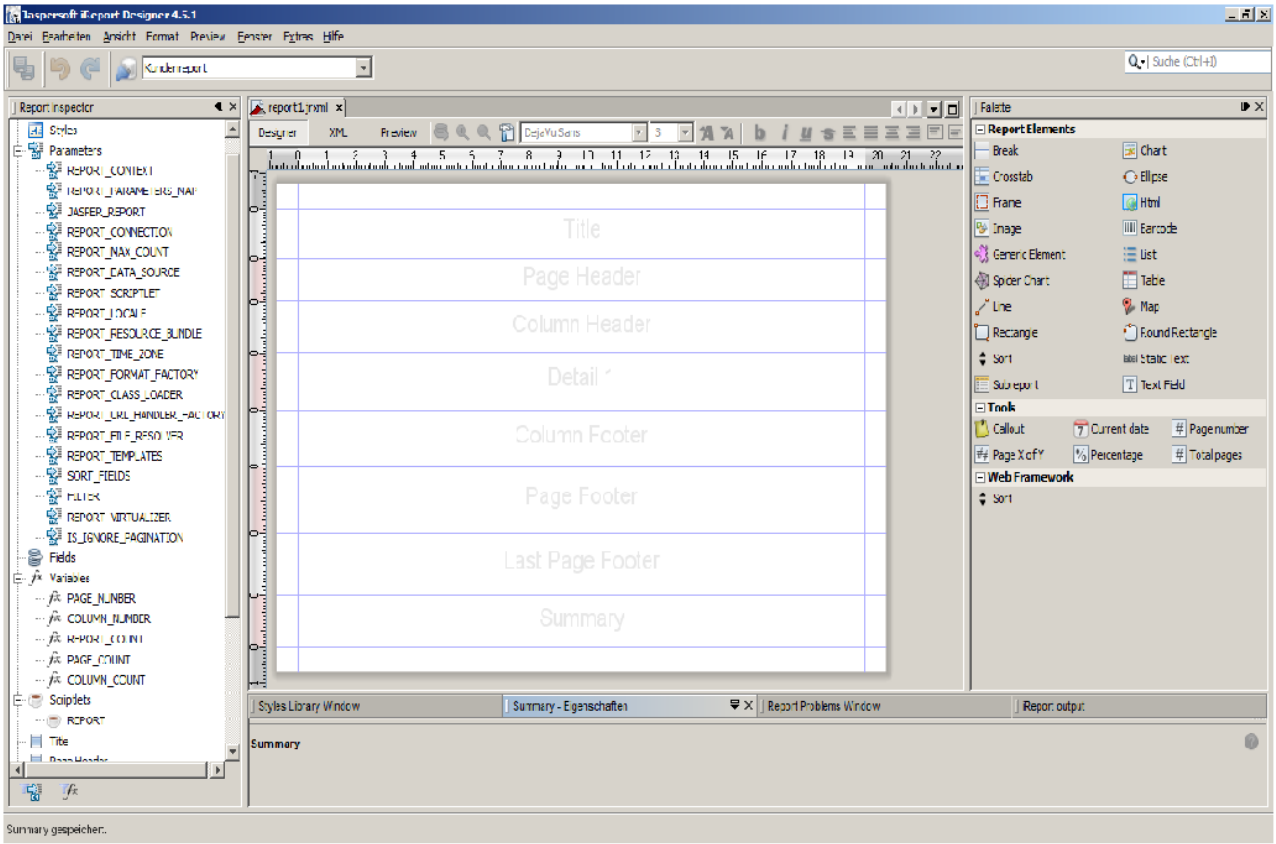

Abbildung 15: iReport-Designer Frontend

IReport-Designer Backend

Report-Designer unterstützt vollständig JRML Syntax. Das bedeutet je nachdem welches Report-Element an welcher Report-Section eingefügt wird, wird automatisch die XML-Struktur dem entsprechend angepasst. Dies ist zwar eine sehr große Unterstützung aus Sicht eines JasperReports-Entwicklers, aber da jedes Tool auch an seine Grenzen stoßen kann, ist es die Aufgabe des Entwicklers die XML-Syntax des Frameworks nicht aus dem Augen zu verlieren. Während der Implementierung und Besonders bei der Umsetzung der Dokumentenvorlagen erforderten eine sehr intensive Einarbeitung in den XML Syntax (DTD und XML Schema)[22] von JasperReports.

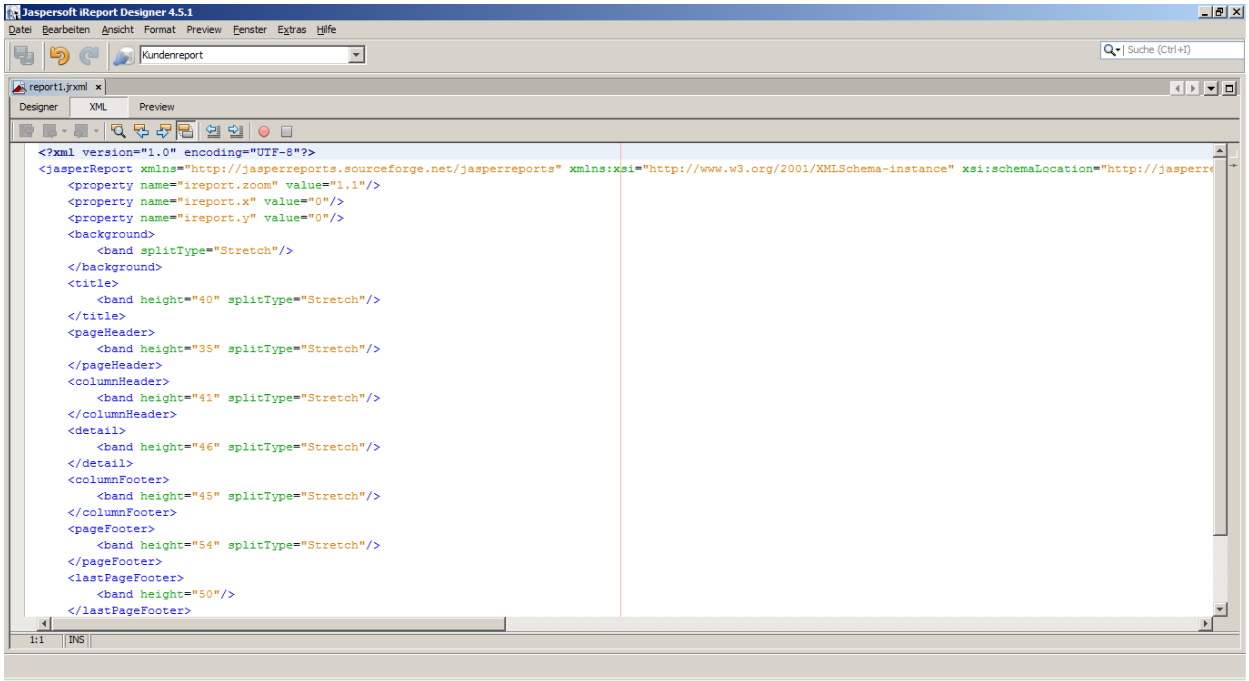

Abbildung 16: iReport-Designer Backend

22 Die Referenzen zu den DTD und XML Schema unter: http://jasperforge.org/uploads/publish/jasperreportswebsite/trunk/schema.reference.html (Abruf: 20.05.2012), http://jasperreports.sourceforge.net/xsd/jasperreport.xsd (Abruf: 20.05.2012) .

Abbildungsverzeichnis

Abbildung 1: Softwarearchitektur....24
Abbildung 2: Aufbau - Softwareprojekt....25
Abbildung 3: BottonUp und TopDown....27
Abbildung 4: Webserviceordner-Struktur....29
Abbildung 5: Webservice Klassen Überblick....30
Abbildung 6: Ablauf Dokumentenerzeugung....31
Abbildung 7: GetReportProxy und GetReportEngine....32
Abbildung 8: Sequenzdiagramm....33
Abbildung 9: GetReportProxy....34
Abbildung 10: GetReportEngine....35
Abbildung 11: Aktivitätsdiagramm....36
Abbildung 12: Musterdesign....38
Abbildung 13: Mustervorlage für Vertragsschreiben....40
Abbildung 14: Mustervorlage für Kundenreport....43
Abbildung 15: iReport-Designer Frontend....50
Abbildung 16: iReport-Designer Backend....51
Abbildung 17: JasperReports Elements....53
Abbildung 18: jasperReports Barcodes....54
Abbildung 19: JasperReports Charts....55
Abbildung 20: Konzept Master-/Subreport....56
Abbildung 21: Webservice - Komponenten Übersicht....57
Abbildung 22: JasperReports - API Overview....58

JasperReports Elements

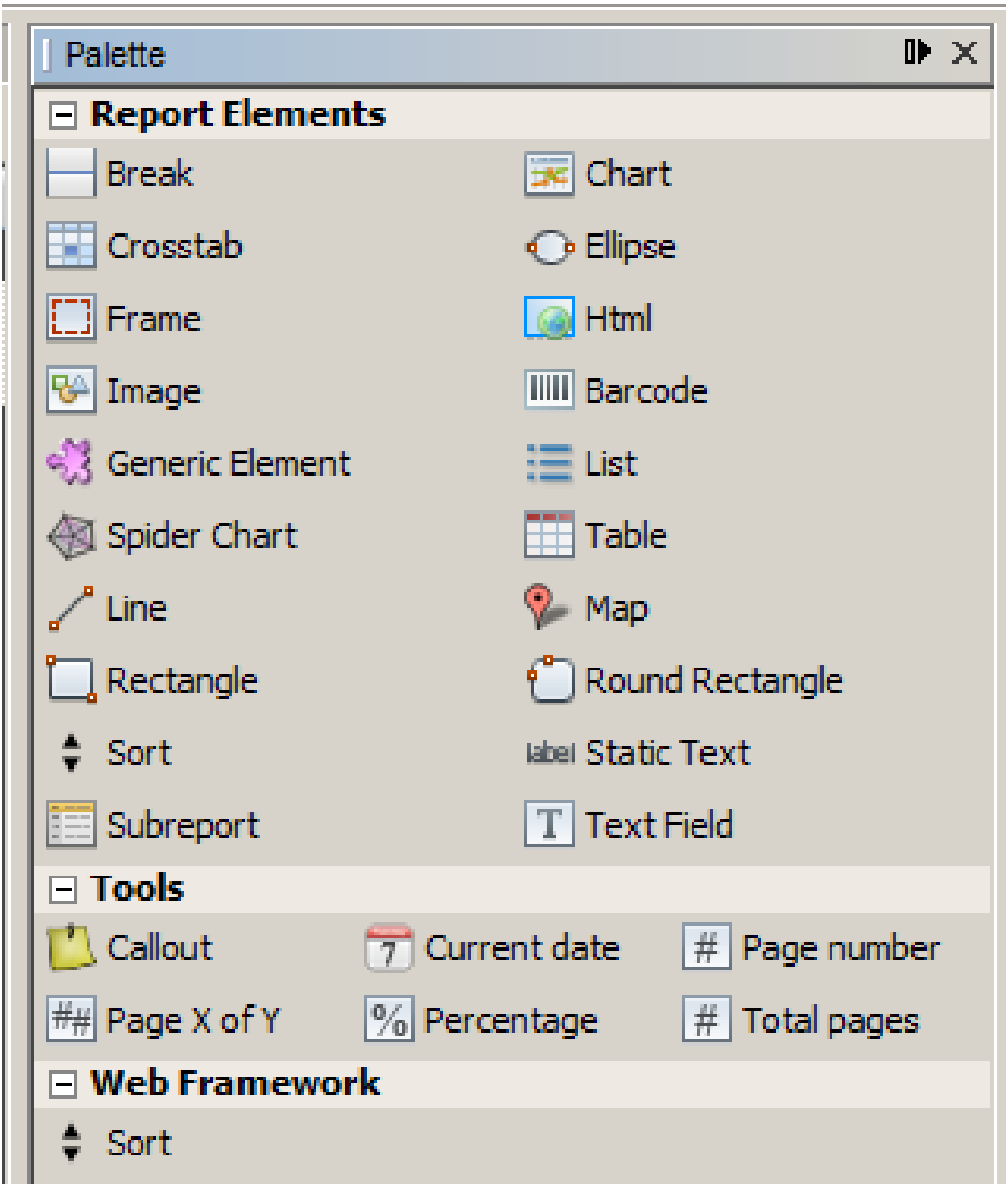

Abbildung 17: JasperReports Elements

JasperReports Barcodes

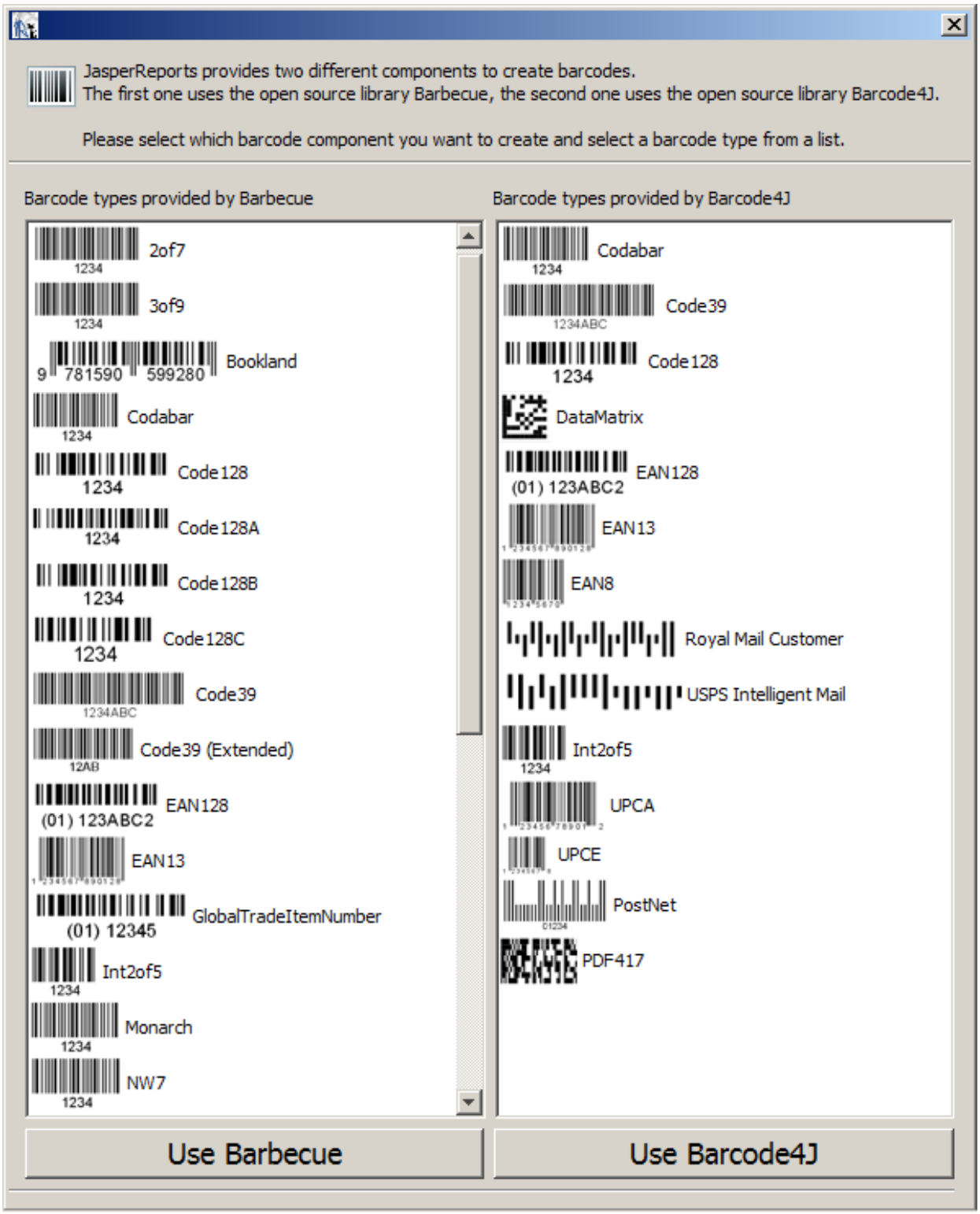

Abbildung 18: jasperReports Barcodes

JasperReports Charts

Abbildung 19: JasperReports Charts

Konzept Master-/Subreport

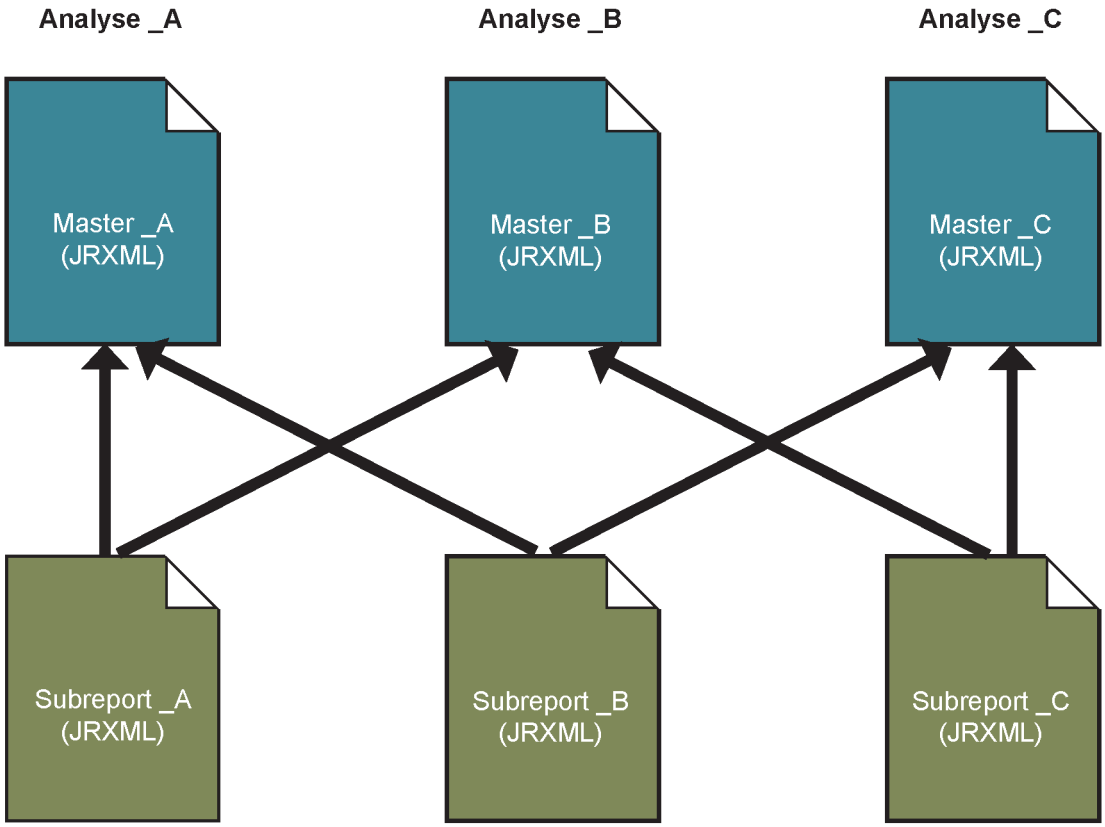

Abbildung 20: Konzept Master-/Subreport

Websrvice – Komponenten Übersicht

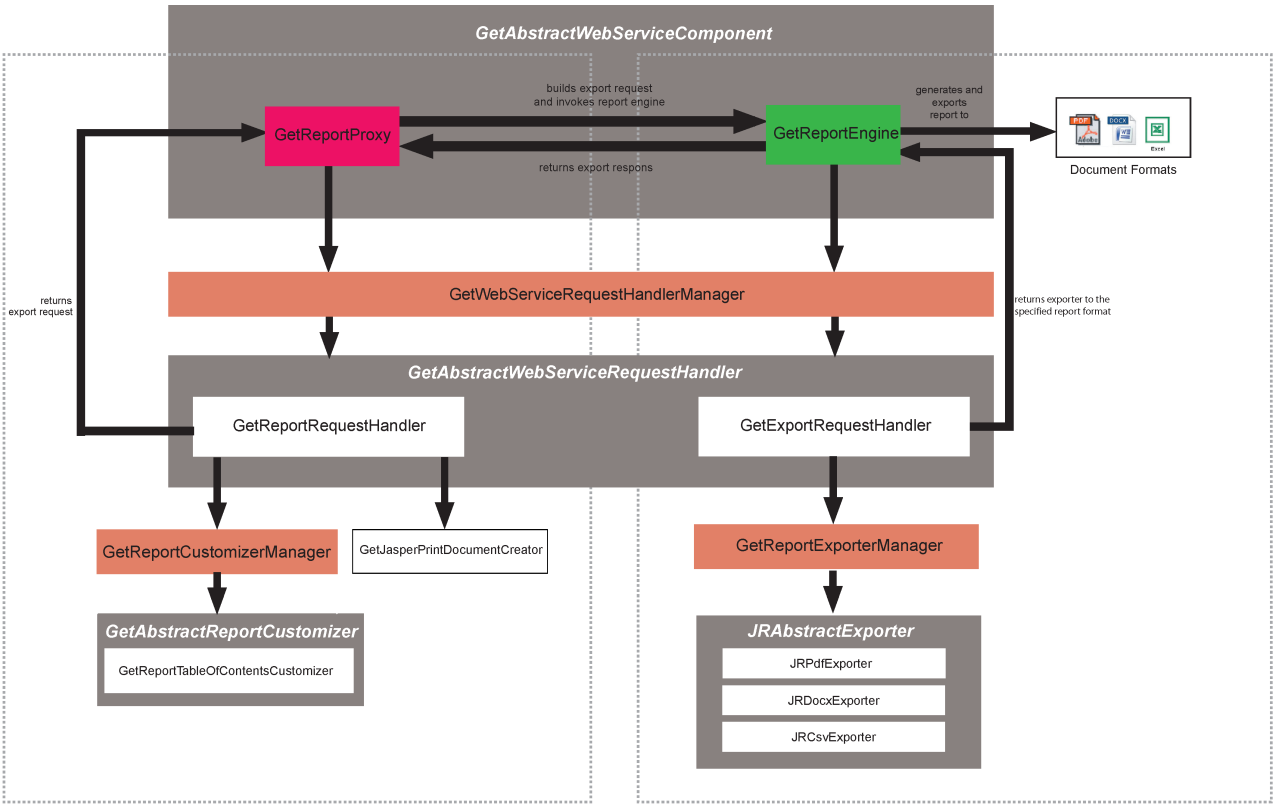

Abbildung 21: Webservice - Komponenten Übersicht

JasperReports - API Overview

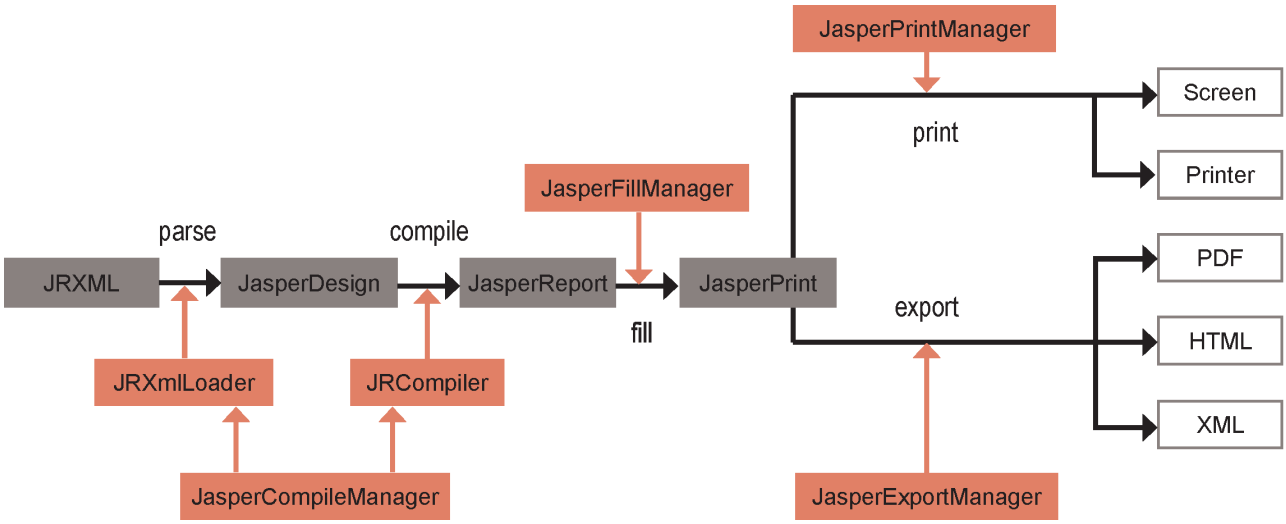

Abbildung 22: JasperReports - API Overview

Quelltextverzeichnis

JasperReports - JRXML Syntax.....47

SubReport Element - JRXML Syntax.....48

GetReportRequest.....61

GetReportResponse.....62

GetExportRequest.....63

GetExportResponse.....64

GetAbstractWebServiceComponent.....65

GetReportProxy.....66

GetReportEngine.....67

GetReportRequestHandler.....68

GetExportRequestHandler.....69

Webservice - WSDL Teil 1.....70

Webservice - WSDL Teil 2.....71

Webservice - WSDD Teil 1.....72

Webservice - WSDD Teil 2.....73

GetReportRequest

```
/**
 * Diese Java-Klasse dient zur Speicherung der Attributte eines
reportRequest-Objekts.
 * Es werden folgende Attribute  gespeichert:
 * masterReport: ist die compilierte JasperReports Master-Vorlage
 * subReport: ist die compilierte Subreport-Vorlage.
 * dataSource: ist Datenquelle. Es handelt ich hier um eine XML-
Document, das alle im Dokument auszugebende Daten enthält.
 * reportFormat: ist das Ausgabeformat (z. B. PDF, DOCX, etc.)
 * @author Mohamed Keita (keita@gmx.net)
 *
 */
public class GetReportRequest implements Serializable {
     private static final long serialVersionUID = 1L;
     // Die Attribute eines GetReportRequest-Objektes
     private String masterReport= null ;
     private String subReport= null ;
     private String dataSource= null ;
     private String reportFormat= null ;
}
```

Quelltext 3: GetReportRequest

GetReportResponse

```
/**
 * Diese Java-Klasse dient zur Speicherung der Attribute eines
ReportResponse-Objekts.
 * Als Ergebnis des Aufrufes des Webservices wird ein
reportResponse-Objekt an den Aufrufer zurueckgegeben - dabei
 * werden reportbinary,
 * reportFormat,
 * reportMessageText
 * und reportMessageCode gespeichert.
 *
 * @author Mohamed Keita (keita@gmx.net)
 *
 */
public class GetReportResponse implements Serializable {
     private static final long serialVersionUID = 1L;
     // Die Attribute eines GetReportResponse-Objekts
     private byte[] reportBinary= null ;
     private String reportFormat= null ;
     private String reportMessageText= null ;
     private Integer reportMessageCode= null ;
}
```

Quelltext 4: GetReportResponse

GetExportRequest

```
/**
 * Diese Java-Klasse dient zur Speicherung der Attributen eines
ExportRequest-Objekts.
 * @author Mohamed Keita (keita@gmx.net)
 *
 */
public class GetExportRequest {
     // Die Attributten eines ExportRequest-Objekt
     private JasperPrint jasperPrintDocument= null ;
     private String reportFormat= null ;
     private String reportEncoding= null;
     private ByteArrayOutputStream reportOutputStream= null;
}
```

Quelltext 5: GetExportRequest

GetExportResponse

```
/**
 * Diese Java-Klasse dient zur Speicherung der Attributten eines
 * ExportResponse Objekt - Dabei werden das gesamte JasperPrint
Document Object des generierten Dokument,
 * das ReportFormat und das ReportOutputStream gespeichert.
 * @author Mohamed Keita (keita@gmx.net)
 *
 */
public class GetExportResponse {
     // Die Attributen eines ExportResponse Objekt.
     private JasperPrint jasperPrintDocument= null;
     private String reportFormat= null;
     private ByteArrayOutputStream outputStream= null;
}
```

Quelltext 6: GetExportResponse

GetAbstractWebServiceComponent

```
/**
 * Diese Java-Klasse ist die Basis-Klasse der Komponenten eines Webservices.
 * @author Mohamed Keita (keita@gmx.net)
 */
public abstract class GetAbstractWebServiceComponent {
      /**
       * Diese Java-Methode soll dazu dienen um der Prozess einer
WebServiceComponent zu implemetieren.
       * @param obj ist das Object was beim Ausfuehren der run-Methode
behandelt wird.
       * @return ein Object, das von der Implementierung abhaengt.
       * @throws GetReportException
       */
      public abstract Object run(Object obj) throws GetReportException ;
      /**
       * Diese Java-Methode soll dazu dienen um ein bestimmtes Objekt zu
verarbeiten.
       * @param obj ist das Object was behandelt wird.
       * @return Ein Object, das von der Implementierung abhaengt.
       * @throws GetReportException
       */
      public abstract Object manage(Object obj) throws GetReportException ;
      /**
       * Diese Mehode leitet ein Objekt zu einem
GetAbstractWebServiceComponent-Objekt und startet seinen Prozess.
       * @param obj ist das Object, was übergeben wird.
       * @param component ist eine Instanz der Klasse
GetAbstractWebServiceComponent.
       * @return Ein Object, das von der Implementierung abhaengt.
       * @throws GetReportException
       */
      public Object dispatch(Object obj, GetAbstractWebServiceComponent
component) throws GetReportException {
            try {return component.run(obj) ;
            } catch (Exception ex) {
                  throw new GetReportException(ex.getCause());}}
}
```

Quelltext 7: GetAbstractWebServiceComponent

GetReportProxy

```
@Override
      public Object run(Object param) throws GetReportException {
            reportProxyLogger.info("[GetReportProxy]: Prozess GetReportProxy
gestartet ...") ;
            GetExportResponse exportResponse= null ;
            try {
                  boolean accepted= false ;
                  if (!(accepted= (param != null && (param instanceof
GetReportRequest)))) {
                        throw new GetReportException(new Throwable(
                                    new String("[GetReportProxy]: Ihre Anfrage
wird vom System nicht unterstuetzt"))) ;
                  }
                  // type cast erfolderlich
                  GetReportRequest reportRequest= null;
                  if (accepted) {
                        reportRequest= (GetReportRequest) param;
                  }
                  if (reportRequest != null) {
                        exportResponse= (GetExportResponse)
dispatch((GetExportRequest) manage(reportRequest),
                                    ((GetAbstractWebServiceComponent)
                                                (new
GetWebServiceComponentManager().loadReportEngine())));
                  }
            } catch (Exception ex) {ex.printStackTrace() ;
                  throw new GetReportException(ex.getCause()) ;
            }
            finally {
                  try {
                        if (exportResponse != null) {exportResponse.flush();
                              reportProxyLogger.info("[GetReportProxy]: Prozess
GetReportProxy erfolgreich durchgefuehrt ...") ;
                        }
                  } catch (Exception e) {
                        throw new GetReportException(e.getCause());}}
            return exportResponse;
      }
```

Quelltext 8: GetReportProxy - run-Methode

GetReportEngine

```
@Override
      public Object run(Object param) throws GetReportException {
            GetExportResponse exportResponse= null ;
            try {
                  reportEngineLogger.info("[GetReportEngine]: Prozess
GetReportengine gestartet ...") ;
                  boolean accepted= false;
                  if (!(accepted= (param != null && (param instanceof
GetExportRequest)))) {
                        throw new GetReportException(new Throwable(
                                    new String("[GetReportEngine]: Ihre Anfrage
wird nicht vom System unterstuetzt"))) ;
                  }
                  // Das Objekt param wird in ein GetExportRequest gecast
                  GetExportRequest exportRequest= null;
                  if (accepted) {
                        exportRequest= (GetExportRequest) param;
                  }
                  /*
                   * Mit Hilfe des ExportRequestHanlders wird der Request
verarbeitet und ein entsprechendes
                   * exportResponse Objekt zurueck.
                   */
                  if (exportRequest != null) {
                        exportResponse= (GetExportResponse)
manage(exportRequest);
                  }
            } catch (Exception ex) {
                  ex.printStackTrace() ;
                  throw new GetReportException(ex.getCause()) ;
            }
            finally {
                  if (exportResponse != null) {exportResponse.flush();
                        reportEngineLogger.info("[GetEngine]: Prozess
GetReportengine Erfolgreich durchgefuehrt ...");}}
            return exportResponse;
      }
```

Quelltext 9: GetReportEngine - run-Methode

GetReportRequestHandler

```
public Object handle(Object param) throws GetReportException {
            reportRequestHandlerLogger.info("[GetReportRequestHandler]:
ReportRequestHandler gestartet ...") ;
            GetExportRequest exportRequest= null ;
            JasperPrint jasperPrintDocument= null ;
            Map<String, Object> reportParametersMap= null;
            GetReportRequest reportRequest= null;
            String masterReport= null;String reportFormat= null;
            try {if (!((reportRequest= inspect(param)) != null)) {
                        throw new GetReportException(new Throwable(
                                    new String("[GetReportRequestHandler]:
Diese Art von Anfrage wird nicht vom System unterstuetzt!"))) ;
                  }
                  masterReport= reportRequest.getMasterReport();
                  reportFormat= reportRequest.getReportFormat();
                  reportParametersMap= new HashMap<String, Object>();
                  putAllReportResourceParametersToMap(reportRequest,
reportParametersMap);
                  jasperPrintDocument= (new
GetJasperPrintDocumentCreator()).createJasperPrintDocument(masterReport,
reportParametersMap);if (jasperPrintDocument != null && reportParametersMap !=
null) {
lookUpForUseCustomizer(GetWebServiceResourceParameter.USE_CUSTOMIZER_PROPERTY_
KEY, reportParametersMap, jasperPrintDocument);
                  }if (jasperPrintDocument != null) {
                        exportRequest= new GetExportRequest();}
            } catch (Exception ex) {ex.printStackTrace();
                  throw new GetReportException(ex.getCause());
            }finally {try {if (exportRequest != null) {
                              setExportRequestParameters(exportRequest,
                                          jasperPrintDocument,
                                          reportFormat, new String("UTF-8"),
new ByteArrayOutputStream());
      reportRequestHandlerLogger.info("[GetReportRequestHandler]: ReportRequest
Handler erfolgreich durchgefuehrt!");}} catch (Exception e) {
                        throw new GetReportException(e.getCause());}}
            return exportRequest ;
      }
```

Quelltext 10: GetReportRequestHandler - handle-Methode

GetExportRequestHandler

```
@Override
      public Object handle(Object param) throws GetReportException {
            GetExportResponse exportResponse= null;
            JasperPrint jasperPrintDocument= null;
            String reportFormat= null;
            GetExportRequest exportRequest= null;
            try {
      exportRequestHandlerLogger.info("[GetExportRequestHnadler]:
exportRequest handling gestartet ...") ;if (!(param != null && (param
instanceof GetExportRequest))) {
                        throw new GetReportException(new Throwable(new
String("[GetExportRequestHandler]: Ihre Anfrage wird nicht vom System
unterstuetzt!")));}
                  exportRequest= (GetExportRequest) param ;if
(exportRequest != null) {reportFormat= exportRequest.getReportFormat();}
                  boolean accepted= false;if (!(accepted= ((new
GetReportExporterManager()).hasResource(reportFormat)))) {
                        throw new GetReportException(new Throwable(
                                    new String("[GetExportRequestHandler]:
Enstprechende Exporter konnte vom System nicht gefunden werden!"))) ;
                  }if (accepted) {jasperPrintDocument=
exportRequest.getJasperPrintDocument();}
                  if (jasperPrintDocument != null) {
      exportRequestHandlerLogger.info("[GetExportRequestHandler]: Report
Exporting Process started...");exportResponse= exportReport(exportRequest, (
                                    (JRAbstractExporter) (new
GetReportExporterManager()).loadReportExporter(reportFormat)));
                  }} catch (Exception e) {e.printStackTrace() ;
                  throw new GetReportException(e.getCause()) ;
            }finally {try {if (exportResponse != null) {
                              exportResponse.flush();
      exportRequestHandlerLogger.info("[GetExportRequestHnadler]: export
request handling finished...") ;
                        }} catch (Exception e2) {
                        throw new GetReportException(e2.getCause());}}
return exportResponse;
      }
```

Quelltext 11: GetExportRequestHandler - handle-Methode

Webservice – WSDL Teil 1

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://jasper.service.ampere.de"
xmlns:impl="http://jasper.service.ampere.de"
xmlns:tns1="http://data.service.ampere.de"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4 Built on Apr 22, 2006 (06:55:48
PDT) -->
<wsdl:types>
<schema elementFormDefault="qualified"
targetNamespace="http://jasper.service.ampere.de"
xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://data.service.ampere.de" />
<element name="getReport">
      <complexType><sequence>
                  <element name="reportRequest" type="tns1:GetReportRequest" /
></sequence></complexType>
</element><element name="getReportResponse">
      <complexType><sequence>
                  <element name="getReportReturn"type="tns1:GetReportResponse"
/></sequence></complexType>
</element></schema><schema elementFormDefault="qualified"
targetNamespace="http://data.service.ampere.de"xmlns="http://www.w3.org/2001/
XMLSchema"><complexType name="GetReportRequest">
      <sequence>
            <element name="dataSource" nillable="true" type="xsd:string" />
            <element name="masterReport" nillable="true" type="xsd:string"
/><element name="subReport" nillable="true" type="xsd:string" />
            <element name="reportFormat" nillable="true" type="xsd:string"
/></sequence></complexType>
<complexType name="GetReportResponse"><sequence><element name="reportBinary"
nillable="true" type="xsd:base64Binary" />
            <element name="reportFormat" nillable="true" type="xsd:string"
/><element name="reportMessageCode" nillable="true" type="xsd:int" /><element
name="reportMessageText" nillable="true" type="xsd:string" />
      </sequence></complexType></schema>
</wsdl:types>
```

Quelltext 12: Webservice - WSDL Teil 1

Webservice – WSDL Teil 2

```
<wsdl:message name="getReportRequest">
     <wsdl:part element="impl:getReport" name="parameters">
     </wsdl:part>
</wsdl:message>
<wsdl:message name="getReportResponse">
     <wsdl:part                element="impl:getReportResponse"
name="parameters">
     </wsdl:part>
</wsdl:message>
<wsdl:portType name="GetReport">
     <wsdl:operation name="getReport">
          <wsdl:input            message="impl:getReportRequest"
name="getReportRequest">
          </wsdl:input><wsdl:output
message="impl:getReportResponse"name="getReportResponse"></wsdl
:output>
     </wsdl:operation></wsdl:portType>
<wsdl:bindingname="GetReportSoapBinding"
type="impl:GetReport"><wsdlsoap:binding style="document"
     transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="getReport">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getReportRequest">
<wsdlsoap:bodyuse="literal"/></wsdl:input><wsdl:output
name="getReportResponse"><wsdlsoap:body use="literal" />
</wsdl:output></wsdl:operation></wsdl:binding><wsdl:service
name="GetReportService"><wsdl:portbinding="impl:GetReportSoapBi
nding"name="GetReport"><wsdlsoap:address
location="http://localhost:8080/JasperWS/services/GetReport"
/></wsdl:port></wsdl:service>
</wsdl:definitions>
```

Quelltext 13: Webservice - WSDL Teil 2

Webservice – WSDD Teil 1

```
<ns1:deployment xmlns="http://xml.apache.org/axis/wsdd/"
      xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
xmlns:ns1="http://xml.apache.org/axis/wsdd/"><ns1:globalConfiguration><ns1:para
meter name="sendMultiRefs" value="true" /><ns1:parameter
name="disablePrettyXML" value="true" /><ns1:parameter name="adminPassword"
value="admin" /><ns1:parameter name="dotNetSoapEncFix" value="true"
/><ns1:parameter name="enableNamespacePrefixOptimization"
value="false" /><ns1:parameter name="sendXMLDeclaration" value="true"
/><ns1:parameter name="sendXsiTypes" value="true" /><ns1:parameter
name="attachments.implementation"value="org.apache.axis.attachments.Attachment
sImpl" /><ns1:requestFlow><ns1:handler
type="java:org.apache.axis.handlers.JWSHandler"><ns1:parameter name="scope"
value="session" /> </ns1:handler><ns1:handler
type="java:org.apache.axis.handlers.JWSHandler"><ns1:parameter name="scope"
value="request" /><ns1:parameter name="extension" value=".jwr" />
</ns1:handler></ns1:requestFlow></ns1:globalConfiguration>
<ns1:handler name="URLMapper"
type="java:org.apache.axis.handlers.http.URLMapper" /><ns1:handler
name="LocalResponder"
type="java:org.apache.axis.transport.local.LocalResponder" />
<ns1:handler name="Authenticate"
type="java:org.apache.axis.handlers.SimpleAuthenticationHandler" /><ns1:service
name="GetReport" provider="java:RPC" style="wrapped"
use="literal"><ns2:operation name="getReport"
qname="ns1:getReport"returnQName="ns1:getReportReturn"
returnType="ns1:GetReportResponse"soapAction=""
xmlns:ns1="http://jasper.service.ampere.de"
xmlns:ns2="http://xml.apache.org/axis/wsdd/"><ns2:parameter
qname="ns1:request" type="ns1:GetReportRequest" /></ns2:operation>
<ns1:parameter name="allowedMethods" value="getReport" />
<ns1:parameter name="typeMappingVersion" value="1.2" />
<ns1:parameter name="wsdlPortType" value="GetReport" />
<ns1:parameter name="className" value="de.ampere.service.jasper.GetReport"
/><ns1:parameter name="wsdlServicePort" value="GetReport" /><ns1:parameter
name="schemaQualified" value="http://jasper.service.ampere.de" />
<ns1:parameter name="wsdlTargetNamespace"
value="http://jasper.service.ampere.de" />
<ns1:parameter name="wsdlServiceElement" value="GetReportService" />
```

Quelltext 14: Webservice - WSDD Teil 1

Webservice – WSDD Teil 2

```
      <ns1:typeMapping
deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
encodingStyle="" qname="ns2:GetReportResponse"
serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
type="java:de.ampere.service.data.GetReportResponse"
xmlns:ns2="http://jasper.service.ampere.de"/><ns1:typeMapping
deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
encodingStyle="" qname="ns3:GetReportRequest"
serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
type="java:de.ampere.service.data.GetReportRequest"
xmlns:ns3="http://jasper.service.ampere.de" />
</ns1:service><ns1:service name="AdminService" provider="java:MSG">
<ns1:parameter name="allowedMethods" value="AdminService" />
<ns1:parameter name="enableRemoteAdmin" value="false" />
<ns1:parameter name="className" value="org.apache.axis.utils.Admin" />
<ns1:namespace>http://xml.apache.org/axis/wsdd/</ns1:namespace>
</ns1:service><ns1:service name="Version" provider="java:RPC">
<ns1:parameter name="allowedMethods" value="getVersion" />
<ns1:parameter name="className" value="org.apache.axis.Version" />
</ns1:service><ns1:transport name="http"><ns1:requestFlow>
<ns1:handler type="URLMapper" />
<ns1:handler type="java:org.apache.axis.handlers.http.HTTPAuthHandler" />
</ns1:requestFlow><ns1:parameter name="qs:list"
value="org.apache.axis.transport.http.QSListHandler" />
<ns1:parameter name="qs:wsdl"
value="org.apache.axis.transport.http.QSWSDLHandler" />
<ns1:parameter name="qs.list"
value="org.apache.axis.transport.http.QSListHandler" />
<ns1:parameter name="qs.method"
value="org.apache.axis.transport.http.QSMethodHandler" />
<ns1:parameter name="qs:method"
value="org.apache.axis.transport.http.QSMethodHandler" />
<ns1:parameter name="qs.wsdl"
value="org.apache.axis.transport.http.QSWSDLHandler" />
</ns1:transport><ns1:transport name="local">
<ns1:responseFlow><ns1:handler type="LocalResponder" /></ns1:responseFlow>
</ns1:transport></ns1:deployment>
```

Quelltext 15: Webservice - WSDD Teil 2

CD-ROM

Der beigefügte Datenträger enthält:

- Eine elektronische Version dieser Arbeit als PDF
- Alle für die Bachelorarbeit relevante Quellcodes
- Alle digital verfügbare Quellen.

Literaturverzeichnis

[1] Teodor, Danciu; Lucian, Chirita: The Definitive Guide to JasperReports. (Apress, 2007). http://ebookbrowse.com/definitive-guide-to-jasperreports-pdf-d126094641, Abruf: 14.05.2012. - ISBN-13 (pbk): 978-1-59059-927-3. - ISBN-10 (pbk): 1-59059-927-6

[2] David R., Heffelfinger: JasperReports for Java Developers. Create, Design, Format, and Export Reports with the World's Most Popular Java Reporting Library (Packt, Juli 2006). http://www.infoap.utcluj.ro/bdate/jreports.pdf, Abruf: 14.05.2012. - ISBN 1-904811-90-6. Production Reference: 1180706

[3] David R., Heffelfinger: JasperReports 3.5 for Java Developers. Create, Design, Format, and Export Reports with the World's Most Popular Java Reporting Library (Packt, August 2009). http://ebookbrowse.com/jasperreports-3-5-for-java-developers-pdf-d209286229, Abruf: 15.05.2012. - ISBN 978-1-847198-08-2. Production Reference: 1050809

[4] JasperSoft Ultimate Guide: The JasperReports Ultimate Guide Third Edition. JasperReports Library. Copyright © 2011 Jaspersoft Corporation. Version: 0811-UGJ37-6. http://jasperforge.org/uploads/publish/jasperreportswebsite/trunk/JasperReports-Ultimate-Guide-3.pdf, Abruf: 15.05.2012

[5] JasperSoft Ultimate Guide: The iReport Ultimate Guide. Ireport. Copyright © 2010 Jaspersoft Corporation. Version: 0510-UGI37-4. http://intranet.uds.edu.py:81/biblioteca/Sistemas/iReport-Ultimate-Guide-3.pdf, Abruf: 15.05.2012

[6] Giulio, Toffoli: The Definitive Guide to iReport (Apress, 2007). http://www.campus.fi.unju.edu.ar/courses/SPJ0001320082AP011/document/recursos_jaser-ireport/The_Definitive_Guide_to_iReport_-

_Apress.pdf?cidReq=SPJ0001320082AP011, Abruf: 15.05.2012. - ISBN-13 (pbk): 978-1-59059-928-0. - ISBN-10 (pbk): 1-59059-928-4

[7] Oracle: The Java EE 6 Tutorial. Copyright © 2012, Oracle and/or its affiliates. All rights reserved. PartNo: 821–1841–15 April 2012. http://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf, Abruf: 03.06.2012

[8] Konzept Master-/Subreports: [1], CHAPTER 11: Subreports, S.129-135. [2], CHAPTER 6: Report Layout and Design. Subreports, S.162-168. [3], CHAPTER 6: Report Layout and Design. Subreports, S.173-179. [4], CHAPTER: Subreports, S.161-168. [5], CHAPTER 10: Subreports, S.153-171. [6], CHAPTER 8: Subreports, S.109-125.

[9] Konzept Main-/Subdatasets: [1], CHAPTER 12: Datasets, S.179. [2], CHAPTER: Subdatasets, S.205-212. [3], Chapter 8: Other JasperReports Features. Subdatasets, S.222-229. [4], CHAPTER: Datasets, S.169-171. [5], CHAPTER 12.2: Using Datasets, S.228-235. [6], CHAPTER 14: Subdatasets, S.221-231.

[10] Konzept Internationalization: [1], CHAPTER 16, S.179. [4], S.233. [5], S.319-325. [6], CHAPTER 10, S.191-195.

[11] Konzept Scriptlets: [2], CHAPTER 8, S.194-199. [4], S.229-232. [5], CHAPTER 17, S.261-274. [6], CHAPTER 11, S.197-201