Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Konstruktion einer Software-Architektur für operatives Reporting am Beispiel eines Prozessleitsystems der BMW AG

Diplomarbeit

Leipzig, Mai 2007

vorgelegt von

Hrushchak, Ruslan
ruslan@hrushchak.net

Studiengang Informatik

Fakultät für Mathematik und Informatik

BMW Werk Leipzig

Zusammenfassung

Die vorliegende Arbeit bietet eine Beschreibung eines Architektur-Konzeptes für eine Reporting-Applikation. Im Hintergrund stehen die theoretischen Konzepte zu Software-Entwicklung und Software-Architekturen sowie zu Reporting-Prozessen in Unternehmen. Die entwickelte Applikation ist in der Lage, operative bzw. Ad-hoc-Reporting-Anforderungen eines Produktionsunternehmen abzudecken. Sie basiert auf J2EE-Technologien und verwendet das open-source Framework Eclipse BIRT. Die Arbeit und die Entwicklung sind innerhalb eines Projektes in der BMW Group entstanden und weisen praktische Erfahrungen mit einer real gegebenen Problematik nach.

Widmung

Meinen Eltern, Maria und Andrij, denen ich alles zur verdanken habe,
meiner lieben Frau, Britta, die mich ständig und geduldig unterstützte,
meiner Schwiegermutter, Ilse, die aufopferungsvoll mein Schreiben korrigierte,
meinen Betreuern, dem Herrn Professor Hans-Gert Gräbe von der Universität Leipzig und
der Frau Kerstin Schöniger von der BMW Group, die mir immer eine freundliche und
exzellente fachliche Betreuung und Unterstützung geboten haben, –

vielen herzlichen Dank!

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1. Einordnung, Problemstellung, Ziele, Inhalte und Vorgehensweise der Arbeit

Die vorliegende Arbeit wurde von 10.2006 bis 4.2007 auf der Grundlage eines Projektes in der BMW Group erstellt. Der schriftliche Teil der Arbeit beinhaltet eine Beschreibung der im Projekt entwickelten Applikation für operatives Reporting und der damit verbundenen Software-Architektur. Zusätzlich liefert die Arbeit eine Analyse der zugehörigen Konzepte und Prozesse.

1.1. Problembeschreibung und Projekthintergrund

Der Ausgangspunkt der Arbeit war die folgende Feststellung:

> Das BMW Group Prozessleitsystem (IPS-T) verfügt nur über eine geringe Anzahl von Reports, und diese decken den Informationsbedarf in den Fachabteilungen nicht vollständig ab. Die zur Verfügung stehenden Reports sind fest definiert, Spezialisten aus den Werks-Fachbereichen haben somit keine Möglichkeit, zusätzliche Reports selbst zu definieren.
>
> Da es sich bei IPS-T um eine Kaufsoftware handelt, können neue Reports ausschließlich vom Software-Hersteller implementiert werden. Die dafür nötige Spezifikation wird von den Key-Usern geliefert. Pro neuer Anforderung / Änderung entstehen extra Kosten. Der Roll-Out dieser Reports ist mit dem Release-Zyklus des Gesamtsystems verbunden. Dadurch ergeben sich längere Wartezeiten für die Endanwender. (s. Hrushchak (2006))

Aus diesen Gründen ist im Oktober 2006 in der BMW Group ein Projekt entstanden, das im Rahmen dieser Diplomarbeit realisiert wurde.

1.2. Ziele und Vorgaben

Das Hauptziel dieser Arbeit ist die Konstruktion einer Software-Architektur für Anwendungen mit Reporting-Hintergrund.

Darüber hinaus soll die Applikation für operatives Reporting als Prototyp implementiert werden und auf einer Testumgebung für die Reporting-Zwecke als Pilot-Plattform ausgerollt werden.

Im Projektauftrag (s. Hrushchak (2006)) sind folgende Ziele formuliert:

- Ziel des Projektes ist die Empfehlung und der prototypische Aufbau einer technisch und fachlich einheitlichen Reporting-Lösung für das BMW-Prozessleitsystem, mit der es möglich ist, Reports zu definieren und zu generieren.
- Die Lösung soll die werks- und technologieübergreifende Fachanforderungen zum Thema IPS-T-Reporting abdecken, Standardreports zur Verfügung stellen sowie einem begrenzten Personenkreis die Möglichkeit bieten, selbst Reports zu erstellen.
- Die Lösung soll als Prototyp umgesetzt und auf einem Test-System als Pilot-Plattform ausgerollt werden.

1.3. Vorgehensweise

Das Projekt wurde nach den Richtlinien der BMW Group umgesetzt und beinhaltete die folgenden Schritte:

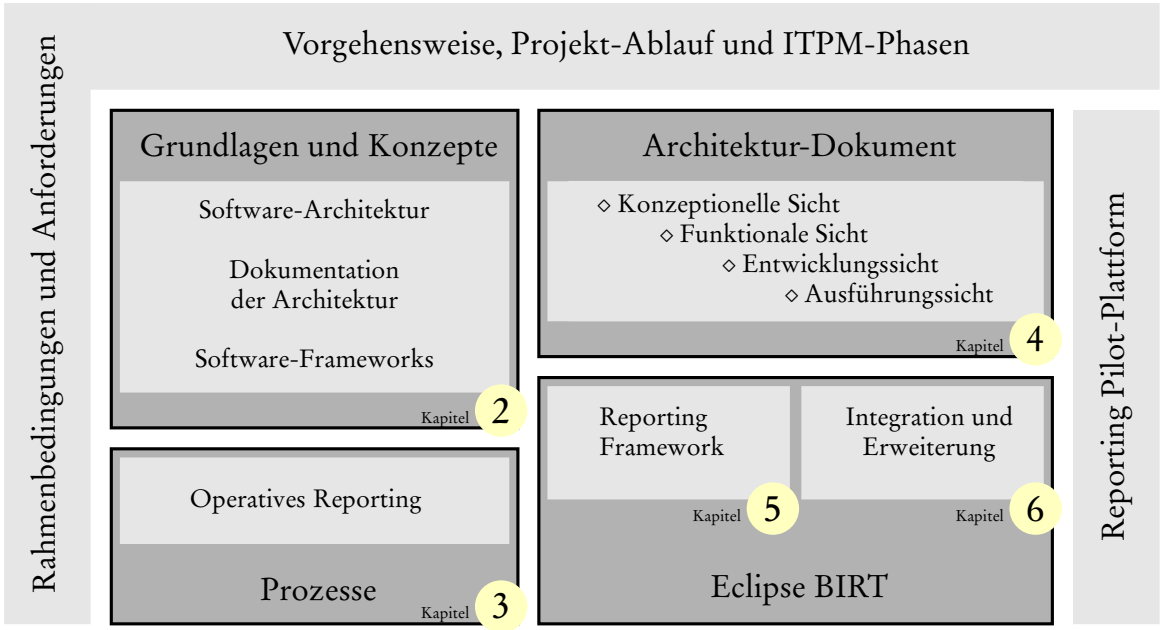
1. Projektauftrag und Marktrecherche
2. Definition der Anforderungen und Erarbeitung des Fachkonzeptes
3. Analyse von Lösungsalternativen und Erarbeitung eines Lösungsvorschlages
4. Entwicklung der IT-Architektur für die genehmigte Lösung
5. Entwicklung der genehmigten Lösung
6. Test und Inbetriebnahme der Pilot-Plattform
7. Schulung der Key-User und Dokumentation

1.4. Inhalt der Diplomarbeit

Die Arbeit beschreibt in erster Linie die Architektur für eine Reporting-Software, dokumentiert die wesentlichen System-Komponenten und analysiert die wichtigsten Entscheidungen.

Nicht alle Projektinhalte und -ergebnisse sind in der Arbeit zu finden. Da der Schwerpunkt im Bereich der Software-Entwicklung liegt, werden vor allem die betriebswirtschaftlichen und Endnutzer-relevanten Fragen des Vorhabens nur am Rande behandelt.

Abbildung 1.1.: Inhalt der Arbeit



Die *Abbildung 1.1* liefert eine schematische Übersicht zum Aufbau der Arbeit.

Die grundlegenden Konzepte zu den Themen Software-Architekturen, Vorgehensweisen und Frameworks sind im Kapitel 2 zu finden. Diese Untersuchungen bilden den theoretischen Hintergrund und die Terminologie-Basis für die vorliegende Arbeit.

Eine Untersuchung der Reporting-Prozesse im Unternehmen sowie die Beschreibung der unterschiedlichen Reporting-Arten und Lösungen sind im Kapitel 3 zu finden.

Kapitel 4 liefert die Beschreibung der Software-Architektur-Sichten und der grundlegenden Design-Entscheidungen.

Die ausführliche Beschreibung der Reporting-Komponente, die auf das Eclipse BIRT-Framework aufsetzt, ist im Kapitel 5 zu finden. Im Anschluss daran werden die Integrationsmöglichkeiten von BIRT sowohl vom architektonischen als auch vom programmiertechnischen Standpunkt aus beschrieben.

Abschließlich werden in Kapitel 7 die Ergebnisse der Arbeit noch einmal zusammengefasst. Es werden außerdem einige Punkte angesprochen, die sich im Rahmen der Arbeit nicht realisieren ließen, sich aber gut fur eine Weiterentwicklung des Systems eignen würden.

2. Software-Architektur – theoretische Hintergründe

Moderne Softwaresysteme sind komplex und umfangreich: Die Nachvollziehbarkeit wichtiger Design-Entscheidungen und die Dokumentation der komplexen Architekturen gewinnen in der Software-Entwicklung immer mehr an Bedeutung.

In diesem Kapitel werden die grundlegenden Konzepte zu den Themen Software-Architekturen, Vorgehensweisen und Frameworks beschrieben. Die Untersuchungen sollen den theoretischen Hintergrund und die Terminologie-Basis für die weitere Arbeit bilden.

Ausgehend von Definitionen der Software-Architektur und der Darstellung von wichtigen Architektur-Konzepten werden Vorgehensweisen für deren Konstruktion im Rahmen eines Projektes beschrieben. Da eine Architektur jeweils in einem unternehmensspezifischen Kontext eingebettet ist, werden zusätzlich auch die ITPM-Phasen vorgestellt.

Eine Software-Architektur wird in Sichten dokumentiert: Somit können die unterschiedlichen Anliegen der einzelnen Stakehoders jeweils als die konzeptionelle, die funktionale, die Entwicklungs- oder die Ausführungs- und Verteilungsicht auf das Software-System berücksichtigt bleiben.

Software-Architekturen und Entwicklungen können mit Hilfe von Frameworks wiederverwendet werden. Die Definition und Beschreibung der Eigenschaften von objektorientierten Frameworks sind im zweiten Teil des Kapitels zu finden.

2.1. Was ist Software-Architektur?

Der Begriff „Software-Architektur“ wird in Fachkreisen oft unterschiedlich definiert.

Der Versuch vom Software Engineering Institute der Carnegie Mellon Universität, den Begriff „Software-Architektur“ exakt zu beschreiben, führte zu mehr als 50 unterschiedlichen Definitionen (Software Engineering Institute, 2007). Martin Fowler bezeichnet solche Architektur-Definitionen als „subjektiv“ und ist fest davon überzeugt, dass mehrere Ausprägungen des Begriffes existieren können. Die Gewichtung von diesem oder jenem Verständnis einer Architektur nimmt seiner Meinung nach in einzelnen Softwareentwicklungsphasen ab oder zu (Fowler u. a., 2002).

Man kann aus der Fachliteratur folgende Thesen entnehmen:

Architektur ist Beschreibung der Struktur
Die Architektur eines Softwaresystems besteht nach Bass u. a. (2003) aus seinen Strukturen, der Zerlegung in Komponenten, deren Schnittstellen und Relationen.

Architekturbeschreibung ist abstrakt
Dynamische und statische Aspekte eines Systems werden als Eigenschaften und Relationen von „Black-Box-Komponenten“ dargestellt. Implementierungsdetails werden bewusst nicht berücksichtigt. Eine Analogie in der Gebäudearchitektur verdeutlicht dies: Eine Grundrisszeichnung enthält üblicherweise keine Informationen über Wasserleitungen.

Architektur ist ein „soziales Konstrukt“
Architektur ist ein „soziales Konstrukt“, das ein gemeinsames Verständnis des Software-Designs in der Entwickler-Gruppe spiegelt. In vielen erfolgreichen Software-Projekten erarbeitet das Entwickler-Team ein gemeinsames Verständnis des Software-Designs und klare Vorstellungen davon, woraus die Software besteht und wie sie aufgebaut wird. Dieses „shared understanding“ wird auch als Software-Architektur bezeichnet (Fowler, 2003). Einigungspunkte sind vor allem Identifikation von Kernkomponenten und Festlegung der Schnittstellen und Methoden für Beziehungen zwischen Kernkomponenten. Software-Architektur beschreibt nur die Komponenten, welche auch von allen Entwicklern verstanden werden müssen. Diese Definition macht klar, dass Software-Architektur ein soziales Konstrukt ist, sie entsteht als Folge einer Einigung in der Entwickler-Gruppe (Fowler, 2003).

Software-Architektur impliziert grundlegende Design-Entscheidungen
Eine Architektur beinhaltet Entscheidungen aus der Anfangszeit eines Software-Projektes. Diese sind anfangs schwer zu treffen und später noch kaum noch zu ändern.

Architektur bildet den Übergang von der Analyse zur Realisierung
Eine Architektur knüpft Verbindungen zwischen der Fachdomäne und deren Umsetzung in der Software, indem sie die geforderte Funktionalität (Anforderungsmanagement) auf eine Struktur von Softwarekomponenten und deren Beziehungen untereinander (Software-Design) abbildet. Siehe Starke (2005).

Architektur besteht aus verschiedenen Sichten
Software-Systeme werden aus unterschiedlichen Perspektiven unterschiedlich wahrgenommen. Für den Anwender ist das System in der täglichen Arbeit unentbehrlich, der Systembetreiber garantiert die tägliche Verfügbarkeit, der Programmierer entwickelt die Software weiter, macht Fehlerkorrekturen oder verwendet einige Systemkomponenten in neuen Projekten weiter. Die wichtigsten Sichten für Software-Architekturen sind die konzeptionelle, die funktionale, die Entwicklungs- und Verteilungssicht sowie die hardwaretechnische und die Betriebssicht. Eine Beschreibung der Sichten ist im *Abschnitt 2.4.4* zu finden.

Software-Architektur wird aufgaben-, domänen- und projektbezogen konstruiert
Gehört etwas zur Architekturbeschreibung, muss dies für alle Entwickler als sehr wichtig erscheinen. Im Fowler (2003) ist folgendes Beispiel zu finden: Spezialisten, die sich mit Enterprise-Applikationen beschäftigen, neigen dazu zu denken, dass Persistenz einer der kritischen Punkte in ihrer Entwicklung ist. Demnach verwenden sie immer für ihre Systeme eine Architektur, die aus drei Schichten besteht. Oft erwähnen sie dabei auch ihre Oracle Datenbank. In dieser Domäne ist das verständlich. Ein Visualisierungssystem in der Medizin dagegen kann durchaus auch eine Oracle Datenbank beinhalten, ohne diese Schicht explizit in der Architektur zu beschreiben. Die Hauptfunktion dieses Systems ist nicht Persistenz, sondern die Analyse von Bildern.

Was ist für eine Software wichtig und besonders wertvoll? Eine entsprechende Architektur soll diese Fragen beantworten können.

2.2. Grundlegende Architekturkonzepte

Moderne Softwaresysteme werden nach dem Schichten-Muster strukturiert und als Multi-Tier-Architekturen gestaltet.

2.2.1. Layers

Layers (Schichten) helfen Systeme, die Elemente mit unterschiedlichen Abstraktionsebenen enthalten, zu strukturieren.

Die Schichten werden nach folgenden Regeln festgelegt:

- Eine Schicht fasst logisch zusammengehörende Komponenten gleicher Abstraktionsebene zusammen.
- Eine Schicht stellt Dienstleistungen zur Verfügung, die an der (oberen) Schnittstelle der Schicht angeboten werden.
- Die Dienstleistungen einer Schicht können nur von Komponenten der direkt darüberliegenden Schicht benutzt werden.

Schichten bauen aufeinander auf. Sie sind nur gekoppelt, wenn sie auch benachbart sind. Nach Buschmann u. a. (1996) dient der Layer-Muster dazu, „eine Anwendung zu strukturieren, die sich in Aufgabengruppen unterteilen lässt, die jeweils eine eigene Abstraktionsebene bilden“.

2.2.2. Tiers

Tiers sind ein Mittel zur Unterteilung eines Systems innerhalb einer Schicht, d. h. innerhalb einer bestimmten Abstraktionsebene. Der Begriff „Tier“ ist orthogonal zum Begriff „Layer“.

Nach dem Patterns-Katalog von Sun (s. Alur u. a. (2003)) existieren fünf Tier-Modelle. Jedes dieser Modelle zeichnet sich aus durch eine klare und einzigartige Zuständigkeit im System[1]. Das ganze System wird durch den Tier-Stapel repräsentiert.

Dabei werden folgende Tiers identifiziert:

Client Tier umfasst die client-seitigen Software-Elemente, die über Desktops und andere Devices dem Benutzer Zugang zu einem Anwendungssystem verschaffen. Die Aufgaben dieser Software-Elemente sind Benutzerinteraktion und grafische Präsentation.

Presentation Tier beinhaltet die Präsentationslogik, das Session Management, die Content Aufbereitung und die Formatierung. Zu ihren Aufgaben gehört die Zugriff-Verwaltung auf die Geschäftslogik – Request, Konstruktion der Antworten und ihre Übermittlung an den Client – Response and Presentation.

Business Tier enthält die fachliche Logik bzw. Geschäftslogik und stellt damit die Business Services bereit, die von den Anwendungsclients benötigt werden.

Integration Tier In dieser Tier werden Komponenten zur Kommunikation mit externen Ressourcen wie z. B. Datenbanken und anderen Systemen zusammengefasst. Die Integration Tier wird oft auch als „Middleware“ bezeichnet.

[1] Das Prinzip der „Trennung von Zuständigkeiten“ (engl. „separation of concerns“).

Resource Tier Diese Tier enthält die Geschäftsdaten. Zur Resource-Tier werden auch externe Ressourcen, wie Nachbar-Systeme und Backend-Systeme gezählt.

Im täglichen IT-Deutsch wird manchmal der Begriff „Schicht“ semantisch falsch verwendet. So spricht man oft z. B. von „Präsentationsschicht“, wobei die „Presentation Tier“ gemeint ist.

2.3. Wie Software-Architekturen entstehen

2.3.1. Einflussfaktoren auf die Software-Architektur

Jede Architektur[2] manifestiert Ergebnisse einer Kette von wirtschaftlichen und technischen Entscheidungen innerhalb einer Organisation. Entwurfsentscheidungen beeinflussen wiederum organisatorische Abläufe. Bass u. a. (2003) bezeichnen diesen Zyklus als „*Architecture Business Cycle*“.

Interessenhalter beeinflussen Architekturen mit Ihren Anforderungen
An der Konstruktion einer Software-Architektur sind meistens unterschiedliche Parteien interessiert: Auftraggeber, Endbenutzer, Entwickler, Projektleiter, Vertriebs- und Betriebsverantwortliche. Diese Personen werden als *Stakeholder* bezeichnet. Software-Architekten haben also die Aufgabe, die Interessen der Interessenhalter zu identifizieren. Dabei besteht die Herausforderung darin, einen Kompromiss zwischen den widersprüchlichen und konkurrierenden Forderungen zu finden.

Tabelle 2.1 zeigt einige solche Faktoren, die im Wettbewerb miteinander stehen.

Tabelle 2.1.: Konkurrierende Einflussfaktoren auf Architektur (Starke, 2005)

Form	⟷	Funktion
Systemanforderungen	⟷	externe Einflussfaktoren
Strikte Kontrolle	⟷	Agile Entscheidungen
Kosten & Termine	⟷	Leistung
Komplexität	⟷	Einfachheit
Neue Technologien	⟷	Etablierte Technologien
Top-Down-Planung	⟷	Bottom-Up-Realisierung
Kontinuierliche Verbesserung	⟷	Produktstabilität
Minimale Schnittstellen	⟷	Enge Integration

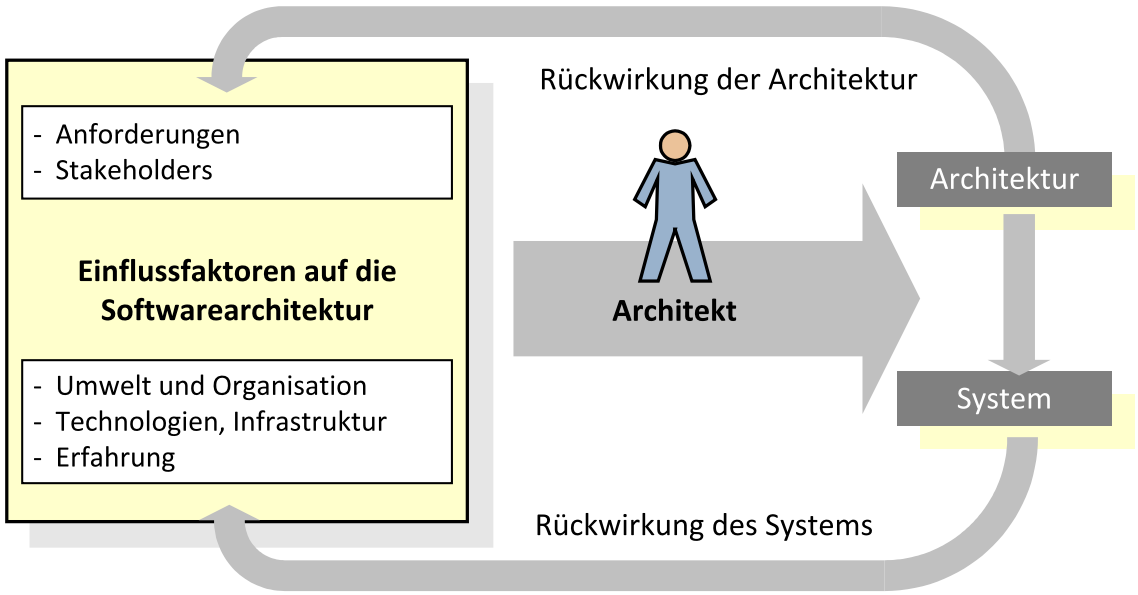
[2] Unter Architektur wird hier und weiter eine Software-Architektur verstanden.

Organisationen beeinflussen Architekturen durch ihre Strukturen
Abhängig davon, welche wirtschaftlichen und strategischen Ziele eine Organisation verfolgt, wie qualifiziert ihre Mitarbeiter sind und wie die entsprechenden Projekte bewertet werden, wird eine Architektur auf diese oder jene Art beeinflusst. Die hochstrategischen Entwicklungen werden dauerhaft investiert und meistens auch als Produktlinien-Architekturen konstruiert.

Erfahrungen und Know-how der Architekten beeinflussen Software-Architekturen
Mit Hang zum technologischen Fortschritt werden einige Entscheidungen mit viel Mut getroffen. Dabei empfiehlt sich, die neuen Technologien durch Prototypen zu erproben.

Architekturen, Organisationen und Systeme beeinflussen sich gegenseitig
Die Rückwirkungen der Architektur und des daraus entstandenen Systems auf die Organisation zeigt die *Abbildung 2.1*. Software-Architekturen haben einen Einfluss auf den Aufbau der einzelnen Entwickler-Gruppen, formieren Unternehmensaufgaben, können System-Anforderungen modifizieren, und in Ausnahmefällen IT-Technologien revolutionieren[3].

Abbildung 2.1.: Architecture Business Cycle. Einflussfaktoren auf die SW-Architektur



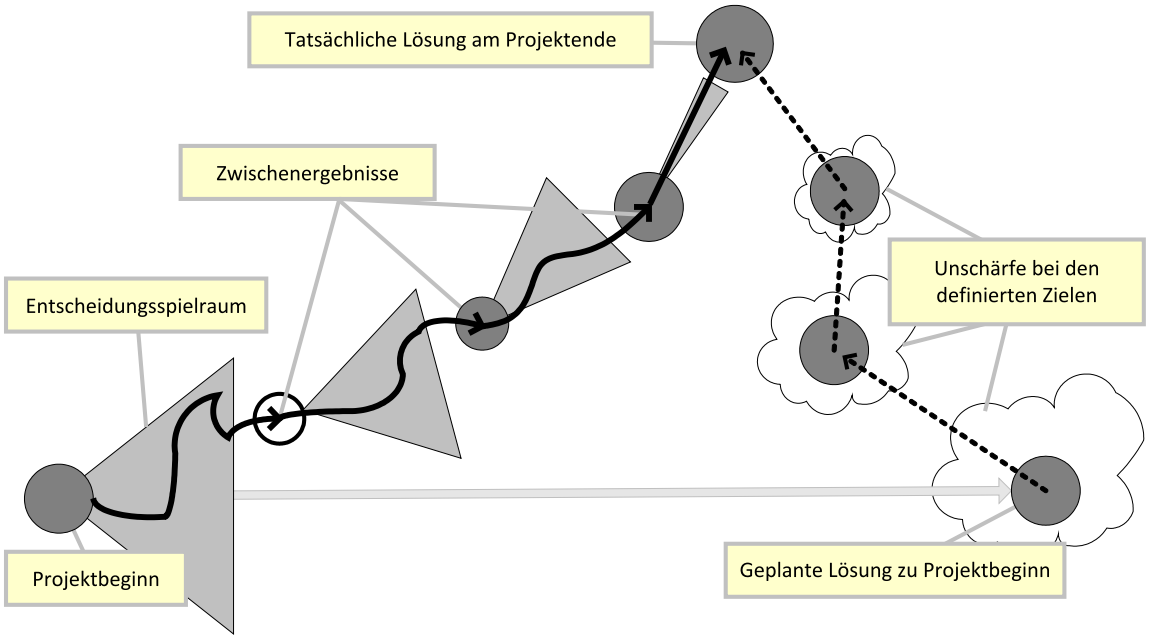
2.3.2. Architektur im Kontext der Softwareentwicklung

[3] 1970 – RDBS, 1980 – GUI, 1990 – WWW, 2000 – J2EE, …

Eine Software-Architektur entsteht in Zyklen und Iterationen

Die Anforderungen an Software, ihre Randbedingungen und Einflussfaktoren ändern sich. Die tatsächliche Lösung am Ende eines Projektes sieht immer anders aus als zu Projektbeginn geplant. In diesem Zusammenhang ist die Metapher der Verfolgung von beweglichen Zielen („moving targets“) sehr gelungen (Starke, 2005). *Abbildung 2.2* zeigt, dass die Zwischenlösungen – Ergebnisse einzelner Iterationen – und (bewegliche) Ziele sich schrittweise immer besser annähern. Gleichzeitig nimmt die Unschärfe bei den definierten Zielen mit dem Projektverlauf ab.

Abbildung 2.2.: Softwareentwicklung ist ein iterativer Prozess.



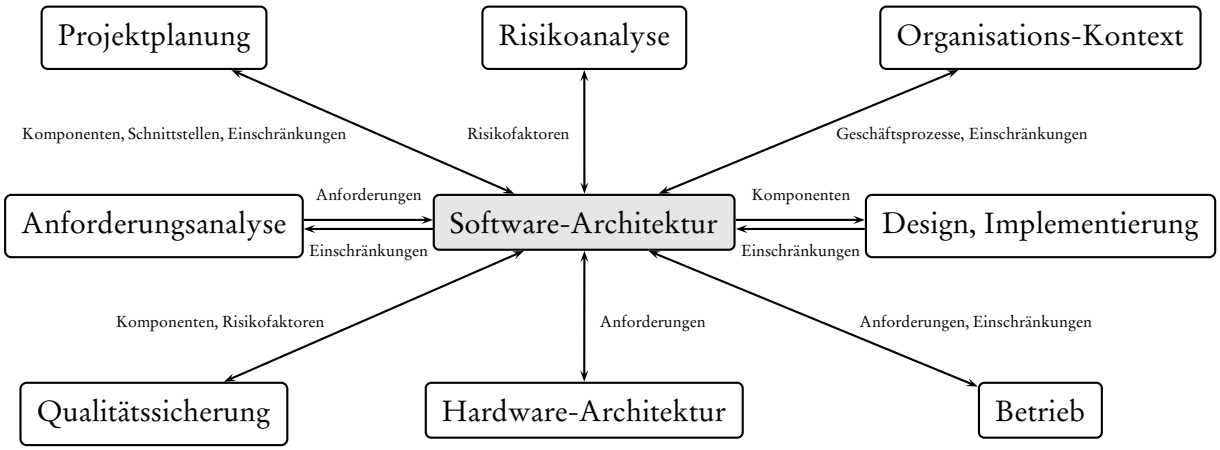
Wie sollte man beim Entwurf von Software-Architekturen vorgehen?

Agile Prozesse, iterative Entwicklung und eine enge Zusammenarbeit mit Stakeholdern sind nach Maier und Rechtin (2000) die wichtigsten Prämissen für eine gelungene Software-Architektur:

> „Verabschieden Sie sich von dem Gedanken, Software-Architekturen ‚einmal für immer‘ zu entwickeln. Entwickeln Sie iterativ. Entwickeln Sie Ihre Entwürfe in Zyklen. Bleiben Sie agil und nehmen Sie Rückkopplungen zu den Projektbeteiligten und der Organisation in Ihre Entwürfe auf. So entstehen Architekturen, die an den wirklichen Bedürfnissen und Anforderungen orientiert sind!“

Architektur im Kontext der gesamten Softwareentwicklung
In Software-Projekten nimmt Architektur eine zentrale Rolle ein. Die *Abbildung 2.3* zeigt, wie Software-Architektur mit anderen Entwicklungsaktivitäten zusammenhängt. Die Prozesse können sich gegenseitig beeinflussen, wenn sie iterativ gefahren werden. Das trägt zur Qualitätsverbesserung der Software bei.

Abbildung 2.3.: Architektur und Softwareentwicklung (Starke, 2005)



Nach Starke (2005) sind bei der Architektur-Konstruktion diese Schritte notwendig:

Informationen sammeln. Informationen über ähnliche Aufgaben und Problemlösungen sammeln. Quellen für Wiederverwendung finden. Eine derartige Recherche kann Organisationsprojekte, Literatur und Internet betreffen.

Systemidee entwickeln. Man sollte am Projekt-Anfang eine genaue Vorstellung wichtiger System-Eigenschaften besitzen. In diesem Schritt sollen folgende Fragen beantwortet werden:

- Was ist die Kernaufgabe des Systems?
- Wie und von wem wird das System genutzt?
- Wie soll die Benutzeroberfläche sein?
- Existieren Schnittstellen zu anderen Systemen?
- Wie werden die System-Daten verwaltet?

Einflussfaktoren und Stakeholder identifizieren. Es gibt organisatorische und technische Einflussfaktoren in Software-Projekten. Organisatorische Faktoren beschreiben Ziele und Erwartungen der Stakeholder. Sie werden dokumentiert und mit Prioritäten versehen. Technische Einflussfaktoren betreffen die Ablaufumgebung des Systems und technische Vorgaben für die Entwicklung, Fremdsoftware, vorhandene Systeme und Referenz-Architekturen.

Wirtschaftliches Modell für das entwickelte System erstellen. Die Kosten-Fragen sind für eine Architektur signifikant. Oft entscheiden sie über Erfolg und Scheitern des Projektes. Wieviel wird die Software kosten? Kann man mit den Investitionen eine Rendite erzielen?

Unterschiedliche Sichten der Architektur entwerfen. Software-Architektur wird in Sichten dokumentiert, um die spezifischen Interessen der verschiedenen Benutzergruppen zu berücksichtigen.

Best Practices – Pattern anwenden. Das Lösen von Problemen ist eine der Hauptaufgaben in der Software-Entwicklung. Untersuchungen zeigen, dass Menschen nicht immer zu den effektivsten Lösungsstrategien im Alleingang kommen, sind aber sehr oft in der Lage, diese zu erlernen und anzuwenden (s. Simon (1996)). Die weisen Software-Architekten lernen bekanntlich nicht nur von eigenen Fehlern. Maier und Rechtin (2000) bringt dies auf den Punkt:

> „Erfolg kommt von Weisheit.
> Weisheit kommt von Erfahrung.
> Erfahrung kommt von Fehlern."

Kenntnisse der „Best Practices" und Fertigkeiten, sie anwenden zu können, kommen nach McConnell (2004) aus der kontinuierlichen Weiterbildung, aus dem Studium und der Analyse anderer Software-Projekte sowie aus der Kommunikation mit Gleichgesinnten.

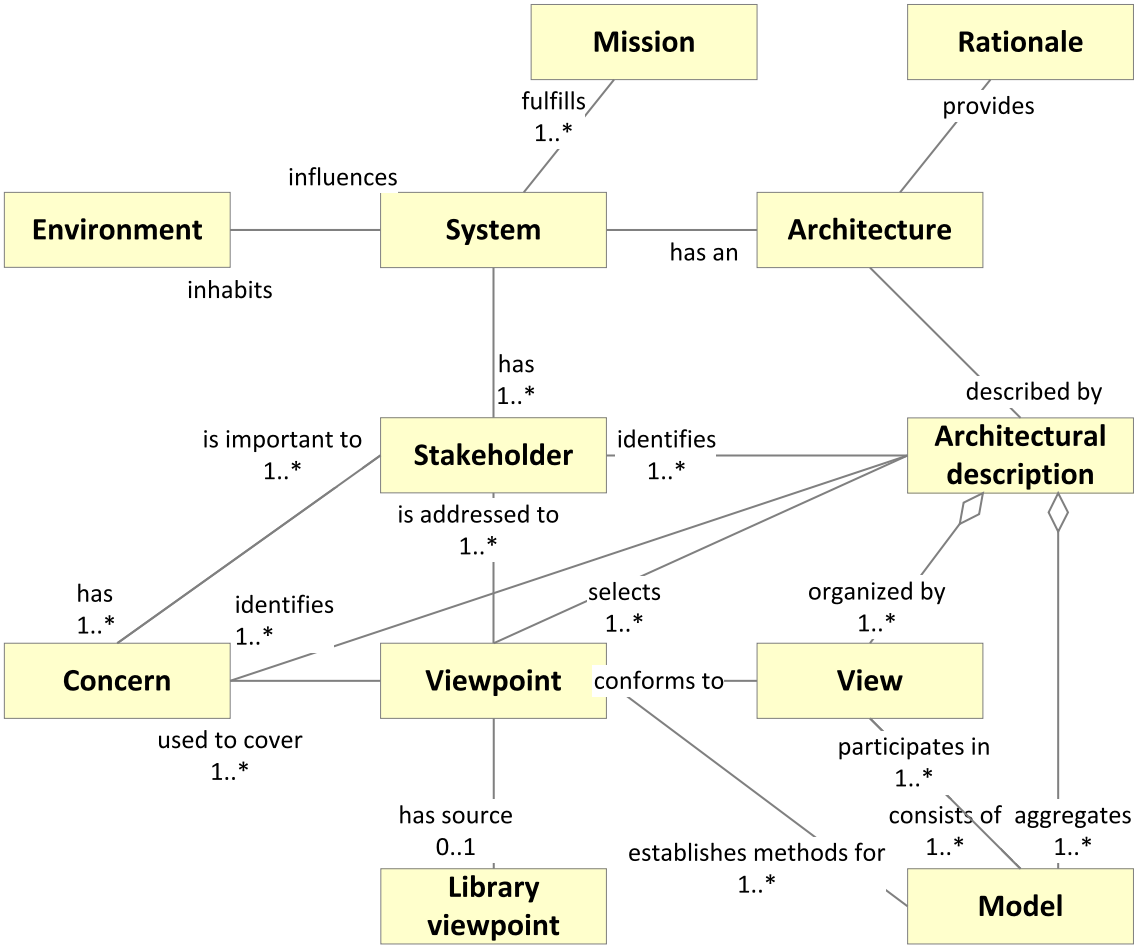
Exzellente Ausführungen der Problematik findet man in Gamma u. a. (1995), Fowler u. a. (2002) und Freeman u. a. (2004).

2.4. Dokumentation der Architektur

2.4.1. Konzeptioneller Rahmen: IEEE 1471

Eine Grundlage für die Beschreibung von IT-Architekturen legt der IEEE-Standard 1471 „IEEE Recommended Practice for Architectural Description of Software-Intensive Systems Description" (IEEE, 2000) fest.

Abbildung 2.4.: IEEE 1471 Conceptual Framework



Der IEEE-Standard 1471 definiert einen konzeptionellen Rahmen für Architekturbeschreibungen. Nach dem IEEE 1471 Basismodell (s. *Abbildung 2.4*) ist ein *System* durch seine *IT-Architektur* beschrieben. Diese Architektur umfasst verschiedene *Sichten* (Views) auf das System bzw. die Architektur des Systems. Diese verschiedenen Sichten adressieren die *Fragestellungen* (Concerns) verschiedener *Stakeholders* – Interessenten und Nutzer der Architektur.

Ausgangspunkt für die Entwicklung einer Architektur sind die Stakeholder und ihre Interessen, die in Anforderungen bzw. Informationsbedarf resultieren. Stakeholder können Organisationseinheiten, wie z. B. Fachbereiche, reale Personen oder Rollen, wie z. B. CIO, Projektleiter, Systemverantwortliche etc. sein.

Eine IT-Architektur wird in einer *Kontext-Umgebung* (Environment), die einen Einfluss auf die Gestaltung des Systems ausübt, konstruiert. Sie umfasst Organisationsstrukturen, Strategien, technische Rahmenbedingungen eines Unternehmens, sowie auch andere Systeme, mit

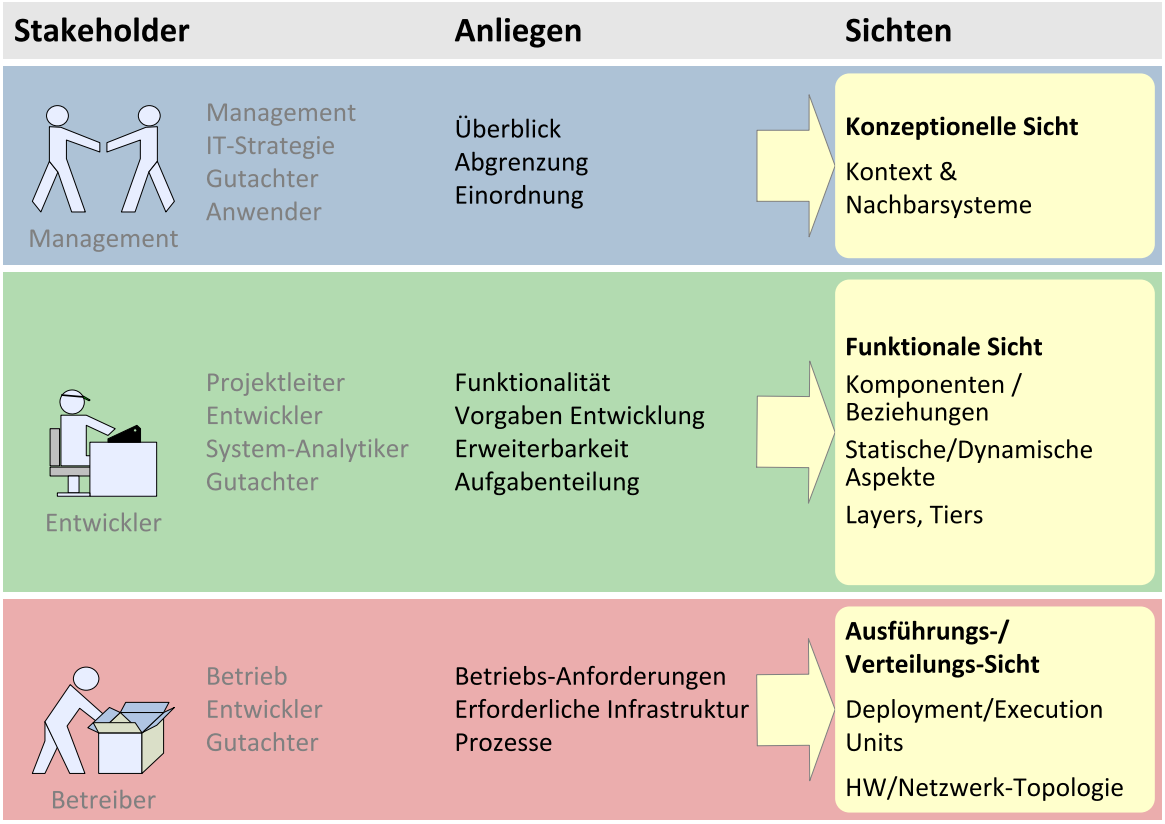
denen das neue System interagieren wird.

2.4.2. IT-Architektur Beschreibung in einem Unternehmen

Aus der konzeptuellen Basis des IEEE-Standards 1471 (s. *Abschnitt 2.4.1*) leitet ein Unternehmen ein eigenes Modell für die Beschreibungen der IT-Architekturen her. Dieses Modell wird dann unternehmensweit für alle IT-Projekte verbindlich.

Bezogen auf das Umfeld eines Unternehmens werden zunächst die Stakeholder und ihre Anliegen analysiert. Aus den gewonnenen Fragestellungen werden dann die einzelnen Sichten auf die IT-Architektur definiert. So benötigt z. B. ein Projektleiter eine andere Sicht auf ein System als z. B. die Verantwortlichen im Betrieb, die das System übernehmen, installieren und betreiben sollen. Die aus einer Analyse resultierenden wesentlichen Views, Stakeholders und Concerns zeigt *Abbildung 2.5*.

Abbildung 2.5.: Analyse der Stakeholder und ihrer Anliegen (nach Böhl u. a. (2003))



2.4.3. IT-Architektur-Dokument

Eine IT Architektur wird durch Grafiken und Text dokumentiert. Neben den Sichten auf das System werden auch Anforderungen und Randbedingungen, welche die Architektur beeinflussen, sowie Architektur- und Entwurfsentscheidungen dokumentiert.

Eine IT-Architektur-Dokumentation kann wie folgend gegliedert werden:

1. Ziele und Vorgaben an das System sowie Randbedingungen
2. Konzeptionelle Sicht (s. Seite 25)
3. Funktionale Sicht (s. Seite 26)
4. Entwicklungssicht (s. Seite 26)
5. Ausführungs- und Verteilungssicht (s. Seite 27)
6. Betriebssicht (s. Seite 28)
7. Übergreifende Konzepte (s. Seite 28)
8. Entwurfsentscheidungen und Varianten (s. Seite 29)

2.4.4. Sichten

Nach IEEE (2000) beschreibt eine Sicht (View) ein konkretes System von einem bestimmten Standpunkt (Perspektive) aus. Eine Sicht abstrahiert von Details, die für die Perspektive nicht von Bedeutung sind.

Dieser Abschnitt bietet eine kurze Beschreibung der einzelnen Sichten auf eine IT-Architektur. Die Ausführungen orientieren sich stark an einen konkreten Unternehmens-Model und Ausarbeitungen von Molterer und Eisele (2003); Böhl u. a. (2003)

Konzeptionelle Sicht

Die konzeptionelle Sicht zeigt das System im Überblick mit den wesentlichen Nachbarsystemen, Schnittstellen und Benutzergruppen. Das betrachtete System wird entweder als Black-Box oder mit den wesentlichen funktionalen Einheiten dargestellt. Durch die Eingliederung wird der Kontext des Systmes verdeutlicht und seine Funktionalität von den Nachbarsystemen abgegrenzt. Die konzeptionelle Sicht beschreibt:

- Die wichtigen Anwendungsfälle und Kernaufgaben: Was soll das System leisten?

- Die wesentlichen funktionalen Einheiten des Systems.
- Das System in seiner Umgebung: Auf welche Nachbarsysteme wird zugegriffen? Wie wird darauf zugegriffen (synchron, asynchron, Datei)? In welche Richtung fließen die Daten?
- Nutzung des Systems: Von wem wird das System genutzt? Welche Benutzergruppen existieren? Wie wird es genutzt (Online, Batch)?

Die konzeptionelle Sicht besteht aus der grafischen Notation und ggf. einer textuellen Beschreibung der Grafik.

Funktionale Sicht

Die funktionale Sicht zeigt die fachlichen und funktionalen Komponenten des Systems. Sie umfasst die Schichten Anwendungssoftware und Software-Infrastruktur und beschreibt die wesentliche Anwendungs- und Softwareinfrastrukturkomponenten sowie ihre Strukturen und Zusammenhänge. Die komplette Funktionalität des Systems wird auf Komponenten abgebildet.

Die funktionale Sicht stellt eine Verfeinerung der konzeptionellen Sicht dar. Die wesentlichen funktionalen Einheiten der konzeptionellen Sicht dienen dabei als Grundlage.

In dieser Sicht werden noch keine Komponenten der Hardware oder Netzwerk-Infrastruktur sowie keine Details der zukünftigen Zielplattform (wie z. B. EJB) gezeigt.

Die funktionale Sicht besteht aus der grafischen Notation und ggf. einer textuellen Beschreibung der Grafik.

Entwicklungssicht

Die Entwicklungssicht zeigt, wie die Komponenten der funktionalen Sicht auf Code-Packages abgebildet werden.

Die Sicht ist nur bei Individual-Entwicklungen sinnvoll. Bei Standardsoftware ist sie überflüssig, weil in diesem Fall die Deployment Units fest vorgegeben sind.

In der Entwicklungssicht wird dargestellt, wie der Source-Code der Komponenten der funktionalen Sicht strukturiert ist und wie aus dieser Struktur die Deployment Units erzeugt werden. Bei kleineren Software-Projekten entspricht die Unterteilung des Systems in Subsysteme bzw. Komponenten oft direkt den Deployment Units.

Die Notation dieser Sicht erfolgt über Tabellen und kann über Diagramme ergänzt werden. Die Strukturierung des Source-Codes kann über „Code Packages“, „Aspekte“, „Namensräume“, Sichtbarkeiten etc. erfolgen. Wenn notwendig werden ebenfalls spezielle Anforderungen an die Entwicklungsumgebung spezifiziert.

Ausführungs- und Verteilungssicht

Die Ausführungs- und Verteilungssicht fasst die Komponenten der funktionalen Sicht zu Deployment Units zusammen und ordnet diese Deployment Units den Hardware-Systemen zu, auf denen diese dann laufen sollen.

Sie beschreibt das System inklusive Infrastruktur zur Laufzeit in einem produktiven Umfeld und dient als Grundlage für die Entwicklung einer Betriebssicht. Dadurch bildet diese Sicht eine „Schnittstelle“ zwischen der Entwicklung und dem Betrieb eines Software-Systems.

Die Ausführungs- und Verteilungssicht kann diese Elemente enthalten:

Hardware-Systeme Hardware-Systeme repräsentieren die Hardware-Infrastruktur, auf der die Anwendung läuft.

Execution Units bzw. Prozesse Diese Einheiten verfügen über ein Lebenszyklus und können gestartet oder gestoppt werden. Somit führen sie zu Prozessen auf den Hardware-Systemen. Execution Units sind z. B. eine direkt ausführbare Anwendung oder der Application Server, in dem die Anwendung deployed wird.

Container Container stellen ihrem „Inhalt“ eine Menge von Diensten bereit. Ein EJB-Container stellt u. a. Sicherheits- und Transaktionsmanagement sowie Namens-Dienste bereit.

Deployment Units Deployment Units sind die verteilbaren Einheiten des Systems. Je nach dem Typ einer Deployment-Einheit können sie nur innerhalb eines bestimmten Containers laufen.

Netzwerkbereiche Netzwerkbereiche oder Sicherheitszonen sind u. a. Intranet, Extranet oder Internet. Die Hardware-Systeme können verschieden Netzwerkbereiche zugeordnet werden.

Die Abbildung der Deployment Units auf die Hardware-Systeme geschieht wie folgt:

- Zusammenfassung der Komponenten der funktionalen Sicht zu Deployment Units
- Wenn nötig Zuordnung der Deployment Units zu Container
- Zuordnung von Deployment Units bzw. Container zu Execution Units
- Zuordnung der Execution Units zu Hardware-Systemen

Im Wesentlichen besteht die Ausführungs- und Verteilungssicht aus der grafischen Notation. Ergänzend erfolgt auch eine textuelle Beschreibung der Grafik.

Hardware-Technische Sicht

Die Hardware-Technische Sicht zeigt den Aufbau der Hardware-Systeme der Ausführungs- und Verteilungsicht.

In vielen Software-Projekten ist diese Sicht optional und wird nur benötigt, wenn spezielle Anforderungen an die Hardware- bzw. Netzwerk-Infrastruktur bestehen, die von der Musterlösung (s. *Abschnitt 2.5.3*) abweichen.

Betriebssicht

Die Betriebssicht zeigt die Instanziierung des Systems in der Betriebsumgebung. Sie beschreibt:

- Die Abbildung der Hardware-Systeme der Ausführungssicht auf konkrete Rechner
- Details zu den verwendeten Hardwareeinheiten und Netzwerkeinheiten und ihre Parameter, wie Rechernamen, IP-Adressen, etc.
- Hinweise zur Software-Installation, wie Pfade, Versionen etc.
- Welche Betreiber (Rollen, Abteilungen) was betreiben und wie die Schnittstellen aussehen

2.4.5. Architekturrelevante, übergreifende Konzepte

Im IT-Architektur-Dokument sollen an einer zentralen Stelle auch die übergreifenden, architekturrelevanten Konzepte beschreiben, welche sich nicht durch die im *Abschnitt 2.4.4* beschrieben Sichten ausdrücken lassen.

Mögliche relevante Themen sind:

Transaktionskonzept Das Transaktionskonzept beschreibt, wie die ACID-Eigenschaft einer Transaktion garantiert wird. Dabei wird beschrieben, welche Komponenten für den „Begin“, das „Commit“ und den „Rollback“ einer Transaktion verantwortlich sind.

Systems Management Systems Management beinhaltet Themen wie Fehlerbehandlung, Logging und Monitoring und beantwortet folgende Fragen:

- Wie wird mit Fehlern umgegangen?
- Welche Informationen werden wie geloggt?
- Wie wird der korrekte Betrieb des Systems überwacht?

Security Security beschreibt, wie die Sicherheitsanforderungen des Systems umgesetzt werden. U. a. werden Themen wie Zugangsschutz (Authentifizierung), Zugriffsschutz (Autorisierung) sowie Verschlüsselung behandelt.

2.4.6. Entwurfsentscheidungen

IT-Architektur-Dokument beinhaltet eine Beschreibung der Entwurfsentscheidungen, die in den einzelnen Sichten gefällt wurden oder die Auswirkungen auf mehrere Sichten haben (z.B. die Auswahl einer Musterlösung oder eines Frameworks).

Die Liste der Entwurfsentscheidungen dient der Nachvollziehbarkeit der Entwicklung der Architektur. Falls die Architektur überarbeitet werden muss, liefert sie Anhaltspunkte, an welchen Stellen Wahlmöglichkeiten bei der Entwicklung der Architektur bestanden.

2.5. Architektur im Projekt

In diesem Abschnitt wird die Vorgehensweise der Entwicklung von Software-Architekturen im Rahmen der Softwareentwicklung-Projekte beschrieben.

Software-Projekte und somit auch die Software-Entwicklungsprozesse werden in vielen Unternehmen oft durch ein angepasstes ITIL-Rahmenwerk[4] (s. ITIL (2006)) und die Prozesse des SLC (Software Life Cycle) beschrieben. Das Projekt im Rahmen dieser Diplomarbeit orientierte sich stark an die Vorgaben von ITPM.

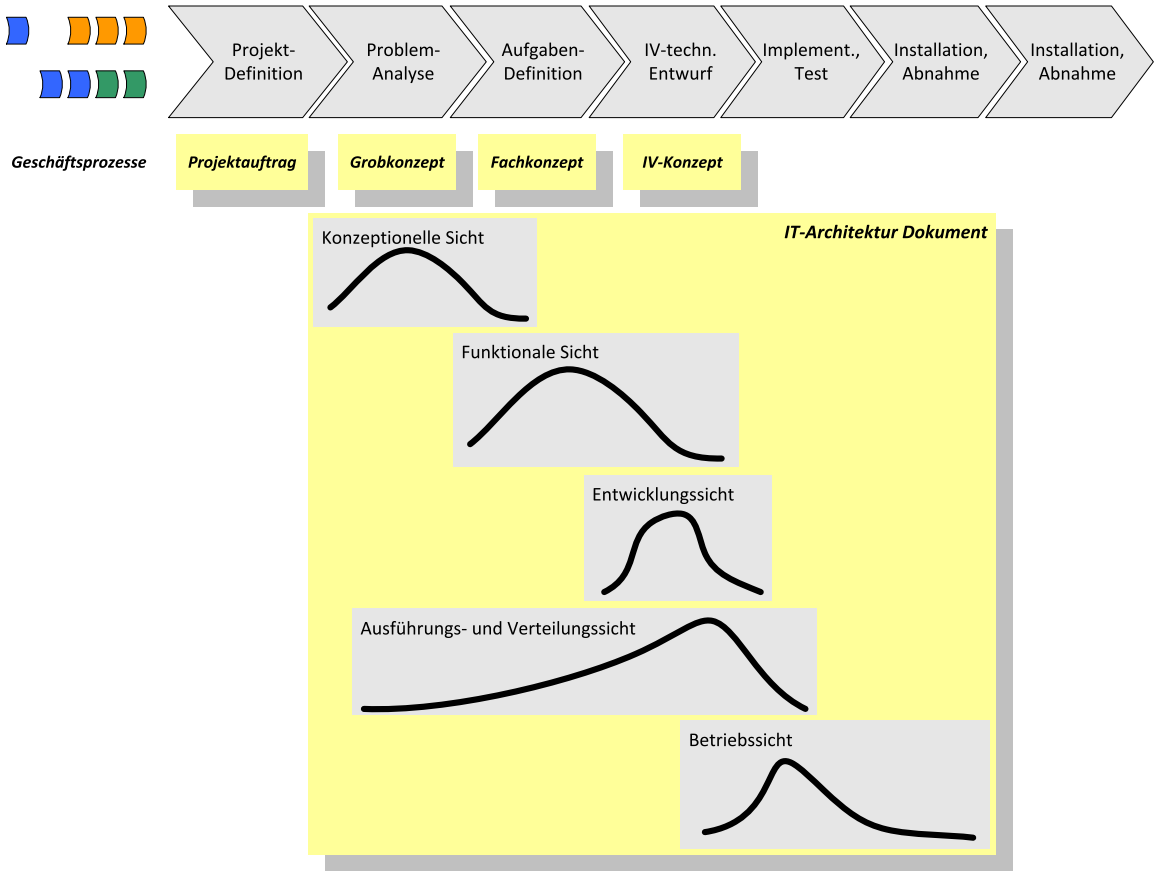
2.5.1. ITPM

IT Prozess Quality Management (ITPM) ist ein BMW-spezifisches Qualitätsmanagement-System für die IT-Prozesse, wie Projektdurchführung, Software-Entwicklung, System-Wartung und Betrieb (s. ITPM (2007); Hrushchak und Priemuth (2006)). ITPM orientiert sich stark an ITIL.

Eine schematische Darstellung der ITPM-Phasen zeigt die *Abbildung 2.6*. Zusätzlich ordnet sie die IT-Architektur-Sichten den einzelnen ITPM-Phasen zu. Die Begrenzung der Kästen deutet an, wann mit der Erstellung der Sicht begonnen wird und wann sie abgeschlossen ist. Die Kurven in den Kästen zeigen, wie sich der Arbeitsaufwand für die Erstellung der Sicht über diesen Zeitraum verteilt.

[4] Information Technology Infrastructure Library

Abbildung 2.6.: ITPM-Phasen (nach Molterer und Eisele (2003))



2.5.2. ITPM-Phasen: Vom Geschäftsprozess zum passenden System

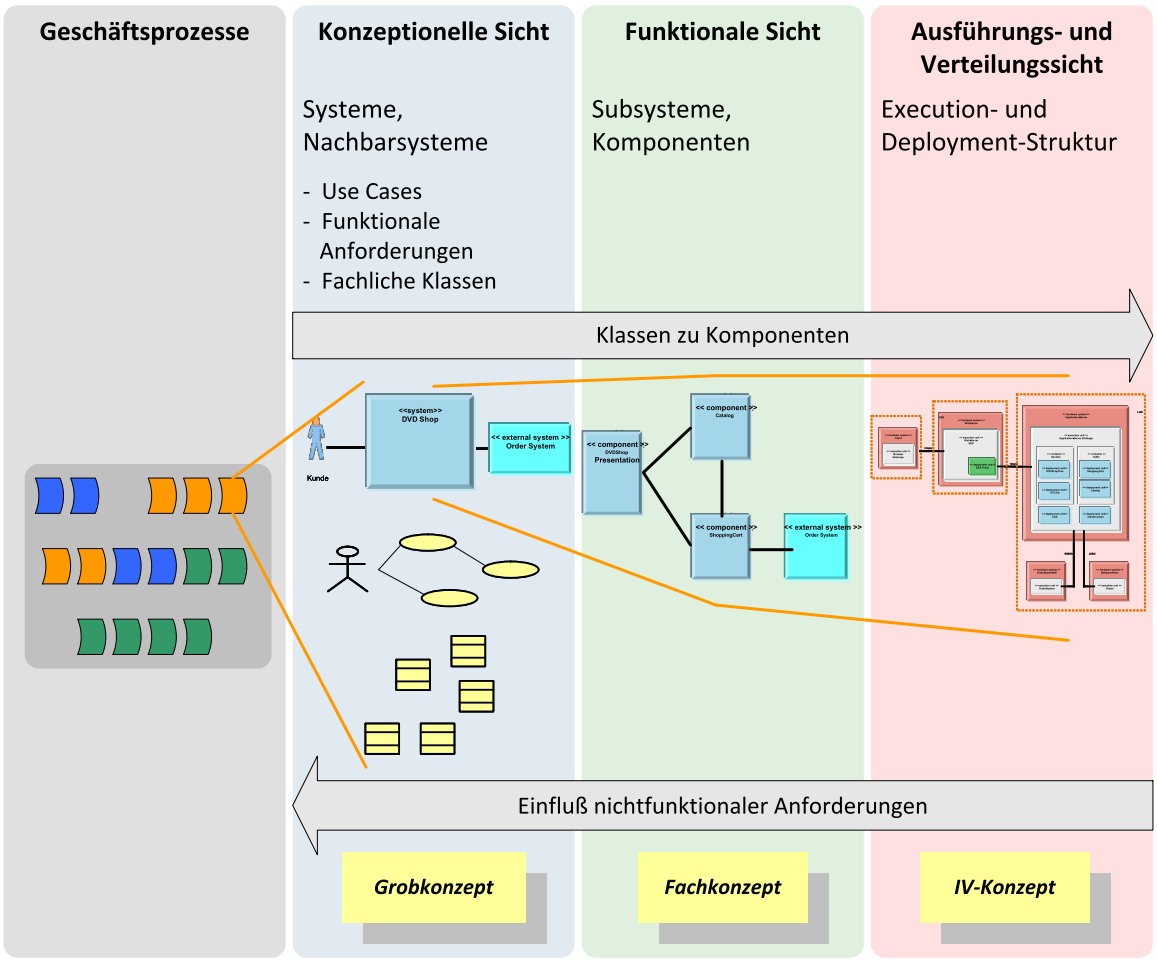
In einem Unternehmen bieten die Anwendungssysteme eine IT-technische Unterstützung für die *Geschäftsprozesse*. Am Anfang der Entwicklung eines jeden Systems steht also der entsprechende Geschäftsprozess im Hintergrund.

Im Folgenden wird erläutert, wie die Inhalte der IT-Architektur in einem ITPM-Projekt zu integrieren sind. Eine Zusammenfassung der Konzepte bietet die *Abbildung 2.7* an.

Im *Grobkonzept* werden zunächst die fachlichen Anforderungen, Use Cases und zentralen Geschäftsobjekte bzw. Fachobjekte aus der Beschreibung des Geschäftsprozesses identifiziert und analysiert. Die konzeptionelle Sicht der IT-Architektur zeigt das System im Gesamtkontext seiner wesentlichen Nachbarsysteme, externen Schnittstellen und Benutzergruppen.

Ausgehend von den Use Cases und fachlichen Szenarien aus der konzeptionellen Sicht, wird im *Fachkonzept* das fachliche Daten- bzw. Objektmodell entwickelt. Die konzeptionelle Sicht wird verfeinert: Das System wird in Subsysteme und Komponenten zerlegt. So entsteht die funktionale Sicht auf die IT-Architektur. Sie wird dann um weitere technische Komponen-

Abbildung 2.7.: Vom Geschäftsprozess zur Architektur



ten erweitert. Dies sind Komponenten der Software-Infrastruktur – allgemeine Querschnitt-Komponenten und Frameworks wie Security-Komponenten, Logging-Dienste, Dialogsteuerung, Präsentationsframeworks und Persistenzframeworks. Das Fachkonzept identifiziert diese Komponenten, eine detailliere Beschreibung dazu liefert das *IV-Konzept*.

Die funktionale Sicht ist produkt- und technologieneutral. Die Abbildung auf produkt- und technikspezifische Komponenten wird in der Ausführungs- und Verteilungssicht erfolgen.

Der dargestellte sequentielle Ansatz (s. *Abbildung 2.7*) vom Geschäftsprozess über die funktionalen Anforderungen und Use Cases zum fachlichen Objektmodell und von den fachlichen Objekten über Komponentenbildung zu Komponenten und Subsystemen entspricht einem *Bottom-Up*-Vorgehen. Dieser Ansatz wird in der Praxis (s. Böhl u. a. (2003)) durch einen *Top-Down*-Ansatz ergänzt. Dabei werden schon frühzeitig die nicht-funktionalen Anforderungen und ihre Auswirkungen auf die Zielarchitektur untersucht. Dies betrifft insbesondere

die Ausführungs- und Verteilungssicht sowie die betrieblichen Aspekte. Diese haben Rückwirkung auf die fachliche und funktionale Komponenten- und Subsystembildung.

2.5.3. Rolle der Musterarchitekturen und Musterlösungen

In einem Software-Projekt unterstützen die Musterarchitekturen und -lösungen das Auffinden der richtigen bzw. passenden IT-Architektur.

Dabei orientiert sich die Entscheidungstabelle an den architekturrelevanten funktionalen Anforderungen, an den Qualitätsanforderungen und an den technischen und organisatorischen Randbedingungen. Anhand von konkreten Fragen kann man zu einer konkreten Musterlösung kommen.

Dabei sind die Musterlösungen zwar verbindlich, sind aber keine Konstrukte „für die Ewigkeit“: Wenn der Bedarf nachvollziehbar ist, kann eine Musterarchitektur in Abstimmung mit verantwortlichen Architekten modifiziert bzw. vervollständigt werden.

Eine IT-Architektur wird in einem Software-Projekt über mehrere ITPM-Phasen konstruiert. Die Prämissen sind der Geschäftsprozess selbst und die fachlichen Anforderungen. Sie richtet sich nach den Vorgaben der Musterarchitekturen und -lösungen des Unternehmens aus.

2.6. Software-Architekturen und Frameworks

Die Wiederverwendung von Software-Architekturen ist ein sehr wichtiger Faktor im Entwicklungsprozess. Dadurch können Entwicklungsaufwand und Entwicklungskosten reduziert sowie Software-Qualität und Zuverlässigkeit erhöht werden.

Software-Wiederverwendung mit Komponenten
Die ursprüngliche Vision der Software-Wiederverwendung basiert auf Komponenten.
Nach Szyperski (2002) ist eine *Software-Komponente*

> „eine Einheit der Komposition mit durch Kontrakt spezifizierten Schnittstellen und nur expliziten Kontext-Abhängigkeiten. Sie kann unabhängig verteilt werden und zur Komposition durch Dritte verwendet werden.“

Frameworks sind keine Komponenten, sie sind komplexer, robuster und können wiederum Komponenten beinhalten. An der Stelle ist eine Abgrenzung von „Komponenten Frameworks“ – einer Komponenten-Menge mit gewissen Gemeinsamkeiten (s. Szyperski (2002)) – angebracht.

Frameworks bieten einen wiederverwendbaren Kontext für Komponenten. Sie bieten eine moderne objektorientierte Wiederverwendungstechnologie an. Langjährige Erfahrungen der

Entwickler und ausgeprägtes Domänen-Wissen werden innerhalb eines Frameworks zu einem einheitlichen Architektur-Gerüst zusammengebracht.

Scherp und Boll (2006) definiert ein Software-Framework folgend:

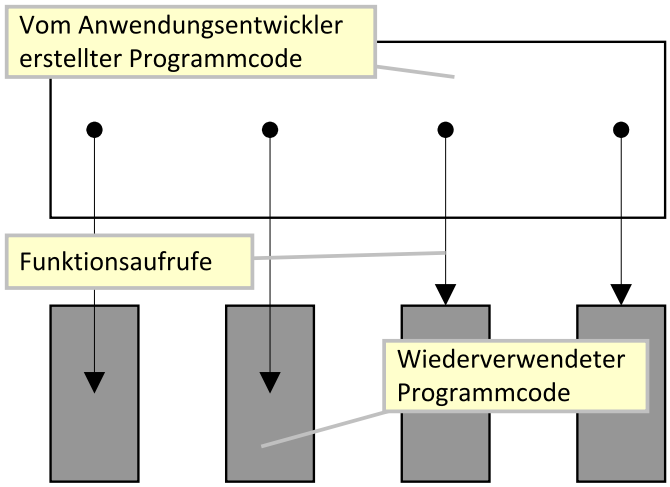
> „Ein Software-Framework stellt typischerweise ein halbfertiges Architektur-Gerüst für einen (komplexen) Anwendungsbereich dar, das auf die Bedürfnisse und Anforderungen einer konkreten Anwendung aus diesem Anwendungsbereich angepasst werden kann."

2.6.1. Eigenschaften von Frameworks

Obwohl Frameworks für unterschiedliche Anwendungs-Domänen wie Telekommunikation, Medizin oder Finanzen entwickelt werden, zeichnen sie sich durch einige gemeinsame Eigenschaften aus. Diese Eigenschaften besagen, was ein Framework ist und wie Frameworks bei der Entwicklung einer Software verwendet werden können. Nach Schmidt u. a. (2004) und Scherp und Boll (2006) besitzen Frameworks folgende gemeinsame Eigenschaften:

1. Umkehrung des Kontrollflusses

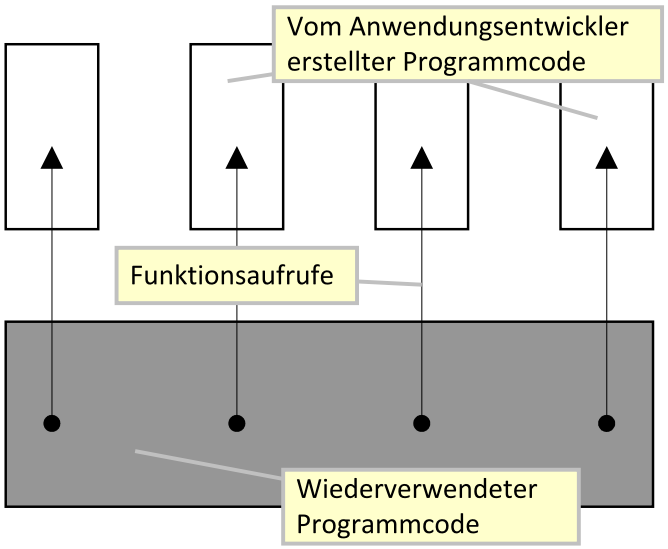
Abbildung 2.8.: Call-Back-Prinzip: Konventionell entwickeltes Anwendungsprogramm (Scherp und Boll, 2006)



Konventionell entwickelte Anwendungen nutzen eine Reihe von (Klassen-) Bibliotheken (APIs). Diese werden von der Anwendung, welche sie nutzt, aufgerufen. Die Bibliotheken sind dabei Hilfsmittel und übernehmen niemals den Hauptkontrollfluss der Anwendung. Dieser Aufruf der externen Klassen wird als Call-Down-Prinzip bezeichnet. Die *Abbildung 2.8* verdeutlicht dies. Immer wenn die Anwendung Funktionalitäten der externen Klassenbibliothek

benötigt, ruft sie diese über festgelegte Schnittstellen auf. Anwendungsarchitektur – Anwendungsstruktur müssen die Entwickler der Anwendung selbst konstruieren, sie wird mit den Klassenbibliotheken nicht mitgeliefert.

Abbildung 2.9.: Call-Down-Prinzip: Auf Framework entwickeltes Anwendungsprogramm (Scherp und Boll, 2006)



Im Gegensatz zu konventionell entwickelten Anwendungen invertieren die Frameworks den Kontrollfluss einer Anwendung. Die Anwendung wird dadurch von Frameworks gesteuert. Die Vorgehensweise ist als Call-Back- oder Hollywood-Prinzip[5] bekannt. Zentrale Bestandteile einer Anwendung und Software-Architektur werden bereits im Framework vorgegeben. Die anwendungsspezifischen Funktionalitäten einer Software, die auf einem Framework aufsetzen, werden meistens als Erweiterungen dieses Frameworks implementiert und werden aus dem Framework heraus aufgerufen (*Abbildung 2.9*). Auf diese Weise wird der Hauptkontrollfluss einer Anwendung durch das Framework gesteuert.

2. Vorgabe einer konkreten Anwendungsarchitektur
Frameworks definieren bereits einen Großteil der Architektur einer Anwendung, halten sie aber an bestimmten Punkten flexibel (Scherp und Boll, 2006). Das sind die Stellen, an denen anwendungsspezifische Funktionalitäten integriert und aufgerufen werden.

3. Anpassbarkeit durch Variationspunkte
Frameworks stellen flexible Variationspunkte zur Verfügung, an denen das Framework an die konkreten Anforderungen einer Anwendung angepasst bzw. erweitert werden kann. Die offe-

[5] „Don't call us, we'll call you!"

nen Variationspunkte werden auch als Erweiterungspunkte bezeichnet.

> „Erweiterungspunkte sind die Stellen in der Framework-Architektur, an denen eine Entscheidung für eine bestimmte Funktionalität zwar bereits typisiert ist, die letztendliche Ausprägung dieser Funktionalität aber offen gelassen und erst durch die jeweilige konkrete Anwendung bestimmt wird.“ Scherp und Boll (2006)

Invertierung des Kontrollflusses und Vorgabe einer konkreten Anwendungsarchitektur machen aus Frameworks generische, erweiterbare, halbfertige Software-Architekturen – „Skelette“. Indem der Entwickler die Erweiterungspunkte eines Frameworks mit anwendungsspezifischer Funktionalität implementiert, werden sie zum fertigen Software-Produkt.

2.6.2. Objektorientierte Frameworks

Historisch bedingt existieren objektorientierte und komponentenbasierte Frameworks.

Komponentenbasierte Frameworks
Komponentenbasierte Frameworks werden ausschließlich durch Komposition von Komponenten angepasst (Szyperski, 2002). Solche Komponenten müssen die vordefinierten Schnittstellen des Frameworks implementieren.

Objektorientierte Frameworks
Im Falle von objektorientierten Frameworks geschieht die Anpassung durch Vererbung und Überschreibung von abstrakten Klassen. Ein objektorientiertes Framework besteht aus einer Menge von konkreten und abstrakten Klassen, die eine generische Lösung für eine spezielle Domäne bereitstellen. Dabei spricht man erstens von einem vollständigen Entwurf – *Frameworks als eine Form der Design-Wiederverwendbarkeit* – und zweitens von einer unvollständigen Implementierung – *Frameworks als „Architektur-Skelette“* (Scherp und Boll, 2006).

Abstrakte Klassen sind die Variationspunkte objektorientierter Frameworks.

Ein objektorientiertes Framework gibt die Anwendungsarchitektur vor. Anwendungsentwickler erweitern ein Framework um gewünschte Funktionalitäten durch das Implementieren der abstrakten Framework-Klassen. Nach dem Call-Back-Prinzip werden diese Klassen vom Framework aufgerufen.

Das Offen/Geschlossen-Prinzip
Grundsätzliche Regel beim Entwurf eines objektorientierten Frameworks ist das Offen/Geschlossen-Prinzip[6] (Meyer, 1997). Es muss offen sein in Bezug auf Erweiterungen und geschlos-

[6] engl. open/close principle

sen hinsichtlich Modifikationen. Die im Framework bereits implementierten Klassen und Methoden dürfen also nicht überschrieben und modifiziert werden.

2.6.3. Frameworks und Softwareentwicklung

Entwicklung von Frameworks
Die Entwicklung eines Frameworks ist eine anspruchsvolle Aufgabe und unterscheidet sich stark von der konventionellen Anwendungsentwicklung. Wenn eine spezielle Anwendung nur einige konkrete Anforderungen erfüllt, sollte ein Framework alle allgemeinen Konzepte einer bestimmten Domäne betrachten und unterstützen.

Nutzung von Frameworks
Scherp und Boll (2006) versteht unter der Nutzung eines Frameworks dessen Einsatz und Anpassung für eine konkrete Anwendung.

Potentielle Probleme mit Frameworks
Abhängig von der Größe und der Anzahl der zur Verfügung stehenden Variationspunkte kann das Erlernen eines Frameworks lange dauern und sehr kompliziert sein (Johnson, 1997). Vor einem Einsatz sollten extern entwickelte Frameworks evaluiert werden. Eine wesentliche Voraussetzung für den erfolgreichen Einsatz von Frameworks ist eine gute Dokumentation. „Framework-Kochbücher“ geben den Entwicklern Auskunft, wie die konkreten Anpassungen und Integration erfolgen können.

Komposition von Frameworks
Moderne Anwendungen versuchen oft mehrere Domänen gleichzeitig abzudecken. Deswegen kommen bei der Entwicklung einer einzigen Anwendung mehrere Frameworks in Frage. Bei der Komposition von Frameworks werden mehrere Frameworks für die Entwicklung einer konkreten Anwendung eingesetzt.

Wenn die einzelnen Frameworks unabhängig voneinander entwickelt wurden, kann eine derartige Komposition zu Problemen führen. So beansprucht jedes objektorientierte Framework die Kontrolle über Applikation für sich alleine (siehe *Abschnitt 2.6.1*) oder die Funktionsumfänge einzelner Frameworks überlappen sich.

Eine Anpassung der Software-Architektur ist in diesem Fall unentbehrlich. Die Interaktion zwischen verschiedenen Frameworks kann auf einer höheren Abstraktionsebene gelöst werden. Nach Scherp und Boll (2006) können so genannte „Frameworks zweiter Ordnung“ dabei helfen, die darunterliegenden komplexen Teilsysteme zu kombinieren und innerhalb einer Anwendung zu integrieren.

Frameworks sind mächtig, aber kompliziert. Die Entwicklung eines Frameworks muss als eine langfristige Investition betrachtet werden. Sie werden in einem iterativen Prozess – Entwicklung, Nutzung, Wartung, (Komposition) etc. – entwickelt.

Den Software-Architekten leisten objektorientierte Technologien wie Komponenten, Patterns und Frameworks eine erhebliche Hilfe beim Konstruieren robuster Systeme.

3. Reporting in Unternehmen

Gegenstand dieses Kapitels ist die Untersuchung der Reporting-Prozesse in Unternehmen.

Die Reports (Berichte) werden im Betrieb sehr oft als Mittel zum Planen, Analysieren und Kontrollieren der Geschäftsprozesse eingesetzt.

Software-Systeme unterstützten den Anwender beim Erstellen und Ausliefern von Reports. Abhängig vom Zweck (wieso bzw. von wem wird ein Report gebraucht) und den Rahmenbedingungen (in welchem Arbeitsumfeld wird ein Report erstellt) entscheidet sich der Anwender für das eine oder andere Reporting-Werkzeug.

In diesem Kapitel wurden zunächst unterschiedliche Reporting-Arten und Lösungen (Kennzahlen, Data Mining, Ad-hoc- und operatives Reporting) analysiert. Daraus wird im nächsten Schritt eine Methodik entwickelt, wie die Reporting-Anforderungen einer passenden Lösung zugeordnet werden können.

Insbesondere die dadurch gewonnenen Reporting-Eigenschaften dienen im Folgenden als Kriterien für die Auswahl eines Reporting-Frameworks und implizieren die organisatorischen Rahmenbedingungen und die funktionalen Anforderungen einer Software-Architektur für operatives Reporting.

3.1. Grundbegriffe

Die Qualität unternehmerischer Entscheidungen wird oft durch die Kenntnis über relevante Daten und Zusammenhänge beeinflusst. Dieses Wissen lässt sich häufig nur durch Analysen erarbeiten, die Daten für solche Analysen liegen allerdings oft bereits im Unternehmen vor und müssen lediglich den Entscheidern zugänglich gemacht werden. Nach Gluchowski (2006) wird dieser Prozess als *Informationslogistik* bezeichnet. Die Aufgaben, „den Mitarbeitern die richtigen Daten zum passenden Zeitpunkt in erforderlicher Form und Genauigkeit am benötigten Platze anbieten zu können", werden durch das betriebliches *Berichtswesen* bzw. Reporting geleistet. Als Synonym wird im Folgenden die Bezeichnung *Reporting* verwendet.

Reports sind Dokumente, die unterschiedliche Daten für einen bestimmten Untersuchungszweck miteinander kombinieren und in aufbereiteter Form vorhalten.

Ein *Reporting-System* liefert eine technische und organisatorische Gesamtlösung zum Erstellen, Ausliefern und Generieren von Reports.

Betriebliche Reports sind Mittel zum Planen, Analysieren, Kontrollieren der Prozesse auf allen Geschäftsebenen. Sie sind an unterschiedliche Nutzer im Betrieb adressiert und werden von Fach- und Führungskräften, Sachbearbeitern, Spezialisten sowie Top-Managern verwendet.

3.2. Reporting – Formalisierung von betrieblichen Informationsangeboten

3.2.1. Formen der Datenhaltung

Operative Daten

In größeren Organisationen und Unternehmen basiert die Abwicklung der Geschäftsprozessen auf einer Vielzahl heterogener Applikationen. Sie werden aufgrund ihrer Orientierung an Transaktionen / Operationen auch als Transaktionssysteme oder operative[1] Systeme bezeichnet. Daten, die solche Systeme erzeugen bzw. generieren, werden *operative Daten* genannt.

Dispositive Daten

In Abgrenzung zu den operativen Daten werden die für managementunterstützende Systeme erforderlichen Daten als dispositive Daten bezeichnet. Die *Tabelle 3.1* verdeutlicht die einzelnen Unterschiede zwischen operativen und dispositiven Daten.

Die operativen Datenbestände eines Unternehmens sind per se sehr umfangreich. Die *Abbildung 3.1* zeigt das Größenverhältnis zwischen den operativen und dispositiven Daten. Diese

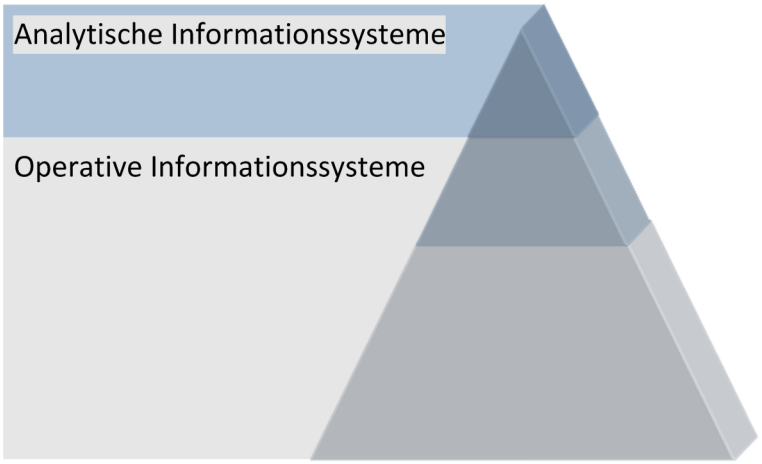
[1] „operativ" (lat.) unmittelbar wirkend

Tabelle 3.1.: Eigenschaften operativer und dispositiver Daten

	Operative Daten	**Dispositive Daten**
Ziel	Abwicklung der Geschäftsprozesse	Informationen für das Management
Datenstrukturen und Modellierung	Transaktions- und funktionsorientiert	Sachgebiets- oder themenbezogen, standardisiert und endbenutzertauglich
Aktualität / Zeitbezug	Aktuell / auf aktuelle Transaktionen bezogen	Historiebetrachtung
Zustand	Redundant, oft inkonsistent	Konsistent modelliert
Verfügbarkeit	Abhängig vom Leistungsprozess (u.U. 7 x 24 - höchste Verfügbarkeit)	partiell

Darstellung wird in der Literatur (Chamoni und Gluchowski, 2006a) als „betriebliche Informationssystempyramide“ bezeichnet.

Abbildung 3.1.: Betriebliche Informationssystempyramide nach (Chamoni und Gluchowski, 2006a)

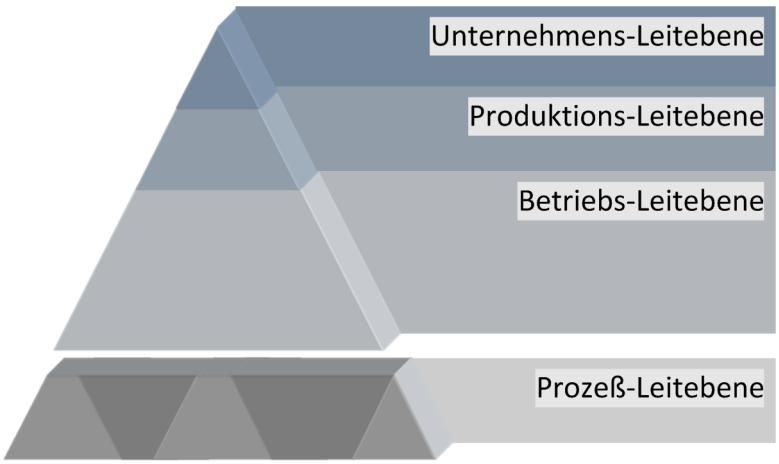


3.2.2. Informationsfluss in Produktionsunternehmen

Um Anforderungen an Informationssysteme in der Produktion besser zu verstehen, ist es notwendig, Modelle des Informationsflusses in Unternehmen zu entwickeln. Bei genauer Untersuchung solcher Informationsflüsse in Produktionsbetrieben stellt man eine Informationshierarchie fest. Nach Polke (2001) ist sie in verschiedene Leitebenen unterteilt (s. *Abbildung 3.2*) und

entspricht oft der Organisationsstruktur eines Unternehmens. Während in der untersten Ebene dieser Struktur eine sehr große Menge von Einzelinformationen über Prozessabläufe im hohen Detaillierungsgrad vorhanden ist, wird sie in den höheren Ebenen bis hin zur Unternehmens-Leitebene immer stärker verdichtet.

Abbildung 3.2.: Informationssystempyramide in Produktionsunternehmen nach (Polke, 2001)



3.2.3. Reporting-Nutzergruppen

Die oben beschriebenen betriebswirtschaftlichen Prämissen implizieren unterschiedliche Reporting-Nutzergruppen (ProClarity, 2005) und demzufolge verschiedene Erwartungen an die Ergebnisse eines Reporting-Systems. Technisch bedeutet dies oft einen Einsatz von unterschiedlichen Software-Systemen.

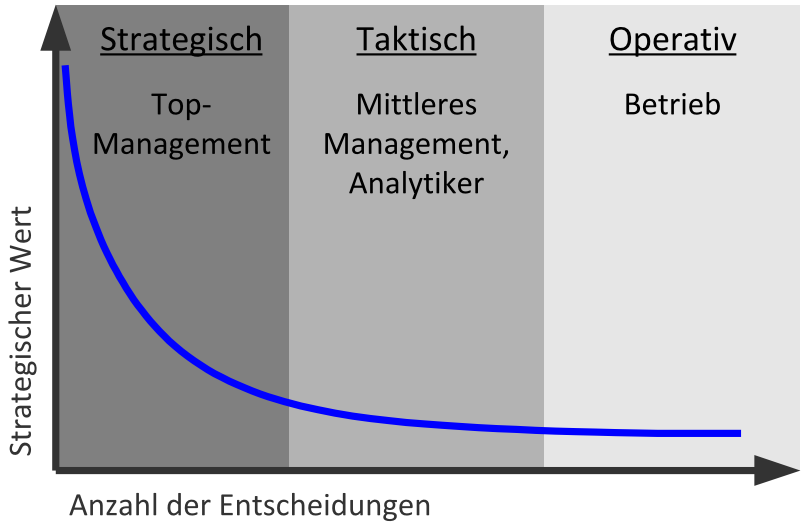
Unternehmensleitung und Manager benutzen betriebliche Daten in einer hoch aggregierten Form als Kennzahlen. Diese Messgrößen bilden eine Grundlage für strategische Entscheidungen und setzen eine festgelegte Berechnungsvorschrift und eindeutige Quellen voraus.

Einige Benutzer – wie Analysten oder Controller – benötigen den betrieblichen Datenbestand lediglich als Ausgangspunkt für eigene, aufwändige Analysen. Sie sollen in der Lage sein, unternehmerische Daten nach interessanten Mustern zu durchsuchen, um aus den Ergebnissen eigene subjektive Verknüpfungen oder Szenarien für das Unternehmen bzw. Prozesse herleiten zu können.

Benutzer im unmittelbaren produktiven Bereich benötigen unternehmerische Daten eher in ihrer Breite und oft in der Roh-Form. Meistens sind die hochaktuellen Detailinformationen zum Fachprozess gefragt. Eine einheitliche Form der Standard-Reports wird an der Stelle vorausgesetzt.

In einem industriellen Report zum Thema „Business Intelligence“ kommen die Berater von

Abbildung 3.3.: Reporting Zielgruppen



Gartner (Friedman und Hostmann, 2004) zum folgenden Fazit:

> „Die Auswahl einer BI-Applikation muss auf der Segmentierung des BI Nutzerkreises basieren. Verschiedene Nutzer besitzen jeweils unterschiedliche Erwartungen an ein BI-Tool. Der gut durchdachte Mix von BI-Tools ist erforderlich, um optimalen Benefit von BI zu bekommen."

3.3. Lösungsszenarien für Reporting-Anwendungen

Basierend auf Literatur und praktischen Projekt-Erfahrungen (s. Bindbeutel (2006)) werden zunächst die Lösungsszenarien für Reporting-Anwendungen diskutiert. Die daraus definierten Reporting-Cluster ermöglichen danach eine strukturierte Analyse der fachlichen Reporting-Anforderungen und ihre Zuordnung zu einem passenden Werkzeug.

3.3.1. Kennzahlen

Kennzahlen sind Messgrößen, die in komprimierter Form Informationen über betriebswirtschaftliche Sachverhalte liefern.

- Für die Ableitung der Kennzahlen werden Daten aus *mehreren Quellen* verwendet.
- Eine Kennzahl ist *hoch aggregiert*, Daten für ihre Berechnung kommen aus mehreren Einzeldatensätzen. Das *Berechnungssystem* bzw. Formeln und die Datenquellen sind klar definiert und eindeutig.

- Der Berechnungsweg ist durch ein *Drill-down* nachvollziehbar: Über eine detaillierte Darstellung ist es möglich zu sehen, wie die Kennzahl berechnet worden ist, welche Werte die Kennzahl beeinflussen etc.
- Die Berechnung wird vom *Administrator* festlegt. Eine Parametrierung kann durch *Benutzer* vorgenommen werden (z. B. Anzahl je Zeitraum).
- Kennzahlen sind an das *(höhere) Management* – keine Experten – adressiert, sie können sowohl lokale (Überwachung der Prozesswerte) als auch unternehmensweite (Zahl der produzierten Fahrzeuge) Größen darstellen. Aus diesem Grund müssen Berechnungsvorschriften *klar definiert und unternehmensweit einheitlich* sein.

Data Warehouse
Wegen Historisierung, Verlässlichkeit, Wiederholbarkeit kommen die Basisdaten für die Kennzahlen in der Regel aus einem Data Warehouse.

Data Warehouses (DWH) sind von den operativen Datenbeständen getrennte, logisch zentralisierte dispositive Datenhaltungssysteme. Sie dienen unternehmensweit als einheitliche und konsistente Basis für alle Arten von Managementunterstützungssystemen (s. Inmon (2002) und Kemper u. a. (2004)).

ETL-Prozess
Der Datenbeschaffungsprozess eines DWH-Systems wird entsprechend seiner Phasen Extraktion, Transformation und Laden als *ETL-Prozess* bezeichnet. Kemper und Finger (2006) beschreiben diesen Prozess folgend:

Extraktion Bei der Extraktion werden die selektierten Daten aus den Strukturen des Quellsystems ausgelesen und in einen temporären Zwischenbereich, die Staging Area, geladen. Die Zeitpunkte der Extraktion werden jeweils fallbezogen festgelegt (z. B. täglich, wöchentlich, monatlich).

Transformation In der Phase der Transformation erfolgt dann eine Bereinigung, Harmonisierung, Aggregation und Anreicherung der extrahierten Daten. Dabei werden u. a. syntaktische und inhaltliche Mängel behoben, die Daten mit gleicher Bedeutung aber mit unterschiedlichen Attributen werden unter einem gleichen Namen abgespeichert (fachliche Abstimmung) und die betriebswirtschaftlichen Kennzahlen werden gebildet.

Laden Den Abschluss des ETL-Prozesses bildet das Laden der Daten in das Data Warehouse.

SAP Business Information Warehouse

Beim *SAP Business Information Warehouse (SAP BW)* handelt es sich um eine Data Warehouse-Lösung der SAP AG. SAP BW beinhaltet u. a. Komponenten für den ETL-Prozess, zur Speicherung von Daten sowie Werkzeuge zur Erstellung von Analysen und Berichten.

In vielen Unternehmen ist diese Lösung bereits im Einsatz. In solchen Fällen wird es daher empfohlen, *die Kennzahlen-Anforderungen mit einer unternehmensweiten Reichweite mit SAP BW zu realisieren*. Analytische Applikationen und SAP BW werden in der Arbeit nur am Rande behandelt, ausführliche Informationen sind u. a. bei Gómez u. a. (2006) zu finden.

3.3.2. Data Mining

> „Knowledge Discovery in Databases (Data Mining) ist ein Ansatz der Datenanalyse und hat das Ziel, in umfangreichen Datenbeständen implizit vorhandenes Wissen zu entdecken und explizit zu machen." (Düsing, 2006)

Data Mining Verfahren erlauben dem Anwender, detaillierte *mathematische und statistische Auswertungen* auf den Daten eines Data Warehouse auszuführen, um *Trends* zu erkennen, *Muster* zu identifizieren und die Daten zu analysieren. Data Mining Eigenschaften werden im Folgenden kurz beschrieben:

- Ein *Fachexperte*, der die Daten beurteilen kann, die „richtige Fragen" darauf stellen und die erhaltenen Ergebnisse interpretieren kann, wäre ein potentieller Nutzer von Data Mining Systemen. Auf diese Weise können z. B. Zusammenhänge zwischen Produktionssparten, Prognosen bei der Einführung eines neuen Produktes oder die Wirksamkeit der Maßnahmen zur Fehlerbehebung ermittelt werden.
- Data Mining Auswertungen werden eher *unregelmäßig* oder auch einmalig durchgeführt.
- Für diese Auswertungen ist ein großer Datenbestand erforderlich, d.h. einerseits viele Quellsysteme und andererseits möglichst detaillierte Daten mit einer weitgehenden Historisierung. Ein *DWH* als Datenquelle eignet sich für diese Zwecke gut.

Data Mining Verfahren und Auswertungen sind nicht der Gegenstand dieser Arbeit. Fundierte Informationen zum Thema sind bei Chamoni und Gluchowski (2006b) zu finden.

3.3.3. Ad-hoc-Reporting

Unter Ad-hoc-Reporting sind interaktive, kurzfristige, *unregelmäßige* oder *einmalige* Abfragen von (detaillierten) Daten, meist aus einem operativen System, gemeint. Ad-hoc-Reporting Eigenschaften sind im Folgenden beschrieben.

- Bei Ad-hoc-Reporting werden kleinere Datenmengen, die meistens aus einer Datenbank kommen, ausgewertet.
- Zielgruppe für Ad-hoc-Reporting sind *Experten eines Fachbereichs*.
- Ad-hoc-Reports werden oft für die Daten-Analyse bzw. Suche im Stör- oder Fehlerfall durchgeführt.
- Die Abfragen und ihre Interpretationen sind *subjektiv*. Eine Analyse der Daten geschieht „im Kopf des Experten".
- Vorgehensweise: Ein Fachexperte definiert interaktiv die Anfrage am Bildschirm, setzt sie ab und interpretiert das Ergebnis für sich. Eine Visualisierung der Daten erfolgt des öfteren mit MS Excel.
- Ad-hoc-Abfragen sind nur sehr *gering wiederverwendbar*.

3.3.4. Operatives Reporting

Operatives Reporting wird durch folgende Eigenschaften definiert:

- Die Reports betreffen nur *einen oder wenige Geschäftsprozesse*, die in einem oder wenigen operativen Systemen durchgeführt werden.
- Die Reports beinhalten *Prozessergebnis-Daten* und *Kennzahlen zu Prozessen*.
- Die Daten sind in der Regel *granular* (detailliert), müssen *kurzfristig* geliefert werden, eine Historisierung von Daten erstreckt sich nur über ein *kurzes Zeitintervall* (z.B. Tag oder Woche).
- Datenquellen sind häufig die *operativen Systeme* selbst oder Operational Data Stores eines Data Warehouses.
- *Anzahl der Quellsysteme* beim operativen Reporting ist immer *gering*, die Datenformate sind homogen, in meisten Fällen sind das relationale Tabellen.
- Zu der Zielgruppe für operatives Reporting gehören *mehrere Nutzer*, die in der Regel keine fachlichen oder technischen Experten sind und aus demselben Personen- bzw. Fach-Kreis kommen.
- Die Abfragen sind *wiederholbar*, und öfter sind dies die gleichen Abfragen mit unterschiedlichen Ausgabe-Darstellungen (z.B. Tabellen oder Diagramme). Eine Analyse – im Sinne interaktiver Suche in den Daten – ist nicht erforderlich.

- Reports bleiben über längeren Zeitraum unverändert, werden *durch einige wenige Experten-Nutzer erstellt und gepflegt*.
- Eine *individuelle Anpassung* der einzelnen Reports ist oft notwendig und wird über *Parametereingabe* seitens des Nutzers gesteuert.
- In der Literatur (s. Gluchowski (2006); Willenborg (2000)) werden operative Reports auch als *formatiertes, vorgefertigtes, parametrisches Reporting* sowie *Online-Auskünfte* oder *Standard-Reporting* bezeichnet.

Wichtige Faktoren für ein erfolgreiches operatives Reporting sind die *Agilität* im Umgang mit dem Reporting-System und die Einbeziehung der Reporting-Konsumenten in den Herstellungs-Prozess (Stichpunkt „*kurze Entscheidungswege*"). Es empfiehlt sich dabei, dass die Reports in Iterationen entworfen und programmiert werden.

3.3.5. Mapping der fachlichen Anforderungen auf eine Reporting-Lösung

Aus den in den *Abschnitten 3.3.1 – 3.3.4* vorgestellten Reporting-Eigenschaften wird im Folgenden ein Fragen-Katalog abgeleitet, der beim Mappen der Reporting-Anordnungen auf die passende Lösungen behilflich sein kann.

Was ist der Verwendungszweck des Reports?

1.	Eine Meldung zum Prozess	⟶	operativer Report
2.	Statistische Analyse, Prognose	⟶	Data Mining
3.	Kennzahlen oder Benchmarking nach einer verbindlichen Formel	⟶	Kennzahlen-System

Wie oft wird Report erstellt?

1.	unregelmäßig	⟶	Ad-hoc-Report
2.	regelmäßig	⟶	operativer Report oder Kennzahlen-System

Ist es abzusehen, dass der Report regelmäßig in der gleichen Form aufgerufen wird?

1.	Ja	⟶	operatives Reporting mit Parametern
2.	Nein, der Report ist eher spontan und wird nur bei gewissen Ereignissen aufgerufen	⟶	Ad-hoc-Report

Ist das Layout des Reports immer gleich?

1.	Ja, der Report soll nach einem standardisierten Layout gestaltet werden	⟶	operatives Reporting oder Kennzahlen-System
2.	Nein, sowohl die Inhalte als auch das Layout des Reports sind oft verschieden	⟶	Ad-hoc-Report

Von wem wird der Report verwendet?

1.	Die Inhalte sind nur lokal (innerhalb eines Fachbereiches) relevant	⟶	operatives Reporting oder Ad-hoc-Report
2.	Die Inhalte sind auch übergreifend relevant und werden vom Management benutzt	⟶	Kennzahlen-System

Wer definiert bzw. erstellt den Report?

1.	Der Experte selbst	⟶	oder Ad-hoc-Report
2.	Der Benutzer verwendet einen vom Administrator vorgefertigten Report und kann ihn über Parameter-Eingabe anpassen	⟶	operatives Reporting oder Kennzahlen-System

Aus wie vielen Quellsystemen kommen die Daten für den Report?

1.	Aus einem Quellsystem	⟶	Ad-hoc-Report
2.	Aus mehreren Quellsystemen	⟶	operatives Reporting
3.	Aus vielen Quellsystemen	⟶	Kennzahlen-System und DWH

Wie aktuell sind die Daten? Bezieht sich die Auswertung auf einen Zeitpunkt oder auf einen Zeitraum?

1.	Für den Report werden die untertägigen Daten (< Tag) benötigt	⟶	operatives Reporting
2.	Der Report wertet Daten über einen längeren Zeitraum aus. Eine Historisierung der Daten ist vorausgesetzt	⟶	Kennzahlen-System und DWH

In welcher Form sollen die Daten vorliegen?

1.	Der Report verwendet detaillierte Daten aus einem operativen System	⟶	operatives Reporting
2.	Der Report verwendet zusammengefasste Daten. Auf die Daten werden u. a. Summen-, Filter-, Gruppen-Funktionen angewendet	⟶	operatives Reporting oder Kennzahlen-System
3.	Der Report verwendet berechnete Daten – Kennzahlen	⟶	Kennzahlen-System und DWH

Die *Tabelle 3.2* bietet eine zusammengefasste Darstellung der Entscheidungskriterien für eine Reporting-Lösung.

Tabelle 3.2.: Kriterien für Auswahl einer Reporting-Lösung (nach Bindbeutel (2006))

Kriterium	Kennzahlen	Data Mining	ad-hoc	op. Reporting
Anzahl der Benutzer	viele	wenige	wenige	viele
Art der Benutzer	(höheres) Management	Experte	Experte	(mittleres) Management, Prozess-Spezialisten
Level der Kennzahl	hoch		mittel	niedrig
Wohin wird berichtet	Unternehmen	lokal/Experte	lokal/Experte	Fachbereich
Zweck	Unternehmens-kennzahlen, Benchmarking	Analyse, Statistik, Prognose	Fehlersuche, lokale Prozess-Steuerung	Fachbereich-kennzahlen, Prozess-Kontrolle
Wer definiert Abfrage	Administrator	User	User	Administrator
Anzahl der Quellsysteme	viele		eins	wenige
Aktualität der Daten / Update-Intervall	lange Historisierung	lange Historisierung	hohe Aktualität	hohe und mittlere Aktualität
Zeitfenster	Zeitraum	Zeitraum	Zeitpunkt	Zeitraum und Zeitpunkt
Granularität der Quelldaten	aggregiert, berechnet	berechnet, Statistik	detailliert	offen
Regelmäßigkeit / Planbarkeit des Reports	ja	nein	nein	ja
Historisierung	langfristig	langfristig	keine	kurzfristig

4. Software-Architektur für operatives Reporting

Dieses Kapitel bildet das Herzstück der Arbeit: Ausgehend von der Theorie der Software-Architekturen aus dem Kapitel 2 und der Reporting-Prozesse aus dem Kapitel 3 wird eine Software-Architektur für das operative Reporting konstruiert und beschrieben.

Zunächst werden Randbedingungen und Anforderungen an die Architektur beschrieben:

- Es werden die projektbezogenen Entscheidungen dokumentiert, die einen Einfluss auf die Entwicklung der Software-Architektur genommen haben.
- Die betroffenen Geschäftsprozesse werden analysiert und die Anforderungen an das System werden dargestellt.
- Zusätzlich werden Antworten auf die Fragen gesucht: Wo ist der Bedarf, den ein Software-System abdecken soll? Soll eine Lösung gekauft oder selbst entwickelt werden? Anhand welcher Kriterien wird eine Entscheidung dazu getroffen?

Die Software-Architektur wird durch die konzeptionelle, die funktionale, die Entwicklungs-, die Ausführungs- und Verteilungsicht auf das Reporting-System beschrieben.

- Die konzeptionelle Sicht beschreibt die Ziele des Systems und die Benutzer-Rollen.
- Die funktionale Sicht unterteilt das System in die Hauptkomponenten und beschreibt die jeweiligen Funktionen.
- Die Entwicklungssicht beantwortet die Frage, wie die Reporting-Applikation entwickelt wird und welche Software-Komponenten zum Einsatz kommen.
- Die Ausführungs- und Verteilungsicht beschreiben das Reporting-System in einer Betriebsumgebung.

4.1. Rahmenbedingungen und Anforderungen

4.1.1. Hintergrundinformation

Das BMW Group *Prozessleitsystem (IPS-T)* hat die Aufgabe der Überwachung von Anlagen im Fertigungsprozess. Das System verfügt nur über eine geringe Anzahl von Reports. Diese decken den Informationsbedarf in den Fachabteilungen nicht vollständig ab.

Da es sich bei IPS-T um eine Kaufsoftware handelt, können neue Reports ausschließlich vom Software-Hersteller implementiert werden. Die Spezifikation wird von den Key-Usern geliefert. Pro neuer Anforderung (Änderung) entstehen extra Kosten. Der Roll-Out von diesen Reports ist mit dem *Release-Zyklus* des Gesamtsystems verbunden. Dadurch ergeben sich längere Wartezeiten für die Endanwender.

Dadurch ergibt sich die Notwendigkeit, eine zusätzliche Reporting-Applikation im IPS-T-Umfeld einzusetzen.

4.1.2. Anforderungen und Prozesse

Wie soll eine Applikation für operatives Reporting aussehen? Welche Funktionen soll sie anbieten? Wie soll sie mit den produktiven Geschäftsprozessen integriert werden?

Antworten auf diese Fragen wurden in Gesprächen mit den End-Usern und in den Reviews mit den Stakeholdern gesucht. Die Ergebnisse sind im Form eines Anforderungskataloges in der *Anlage A* zu finden. Die folgenden Thesen liefern eine kurze Beschreibung der Prozesse:

- Die Reports (Berichte) sind für IPS-T-Anwender ein wichtiges *Mittel zum Planen, Analysieren und Kontrollieren der Geschäftsprozesse*. Sie sollen u. a. tagesaktuelle Daten auflisten (Tabellen), visualisieren (Diagramme) und aggregieren (Kennzahlen) können.
  Eine ausführliche Darstellung der Problematik ist im *Abschnitt 3* zu finden.

- Die Rolle der Report-Ersteller übernehmen die *Key-User* des Systems. Die Rolle der Report-Nutzer (Informationsmanagement) wird vor allem von den Key-Usern, dem mittleren Management und den Werkern übernommen.

- Die Reports werden innerhalb einer Web-Applikation zur Verfügung gestellt. Für das Aufrufen und Generieren von Reports darf keine Lizenzierung erforderlich sein.

- Eine wichtige Prämisse für die IPS-T-Reports sind die operativen Daten des Systems. Das System darf aber nicht auf die Produktions-Datenbank zugreifen, um mögliche Engpässe zu vermeiden.

- Angesicht der vielen abzulösenden Java-Alt-Systeme aus diesem Bereich soll die Lösung eine Technologie-Konsolidierung und Standardisierung bei den Prozessen unterstützen und damit einer Vision der homogenen Systemwelt näher kommen.

4.1.3. Analyse des Ist-Zustandes

Derzeit werden die Reporting-Prozesse in einigen betroffenen Fachabteilungen nach deren Eigeninitiative u. a. mit Microsoft Excel, Cognos Impromptu oder als manuell implementierte JSPs unterstützt. Aus einer Analyse der Tools und Prozesse geht hervor, dass keines dieser Tools den Anforderungen (s. *Abschnitt 4.1.2*) im vollen Umfang entspricht noch das Potential hat, in der vorhandenen Form über alle Werke und Fachabteilungen ausgerollt zu werden.

Reporting mit Microsoft Excel Dies ist oft mit einem zeitintensiven, fehleranfälligen und redundanten Prozess verbunden.

- Datensammlung, Transport und Verwaltung erfolgen manuell: Die benötigten Daten werden oft mit „copy and paste“ aus der IPS-T-Web-Applikation entnommen.
- Datenauswertungen, Reports und Layouts sind nicht standardisiert und im besten Fall nur lokal konsolidiert.
- Unterschiedliche Report-Ergebnisse bzw. Darstellungen sind bei den gleichen Daten wahrscheinlich.

Manuelle Implementierung von Reports Die Reports werden als einzelne JSP-Seiten implementiert.

- Der Prozess ist ressourcenintensiv und erfordert IT-Know-how der Key-User, was nicht immer gegeben ist.
- Solche Applikationen sind über Jahre hinweg entstanden, verwenden jeweils recht unterschiedliche Frameworks und Komponenten und sind somit gering vereinbar oder wiederverwendbar.

Reporting mit Cognos Impromptu Für das Reporting wird das Desktop-basierte Reporting-Tool Impromptu von Cognos[1] eingesetzt.

- Impromptu ist eine lizenzpflichtige Anwendung.

[1] http://www.cognos.com

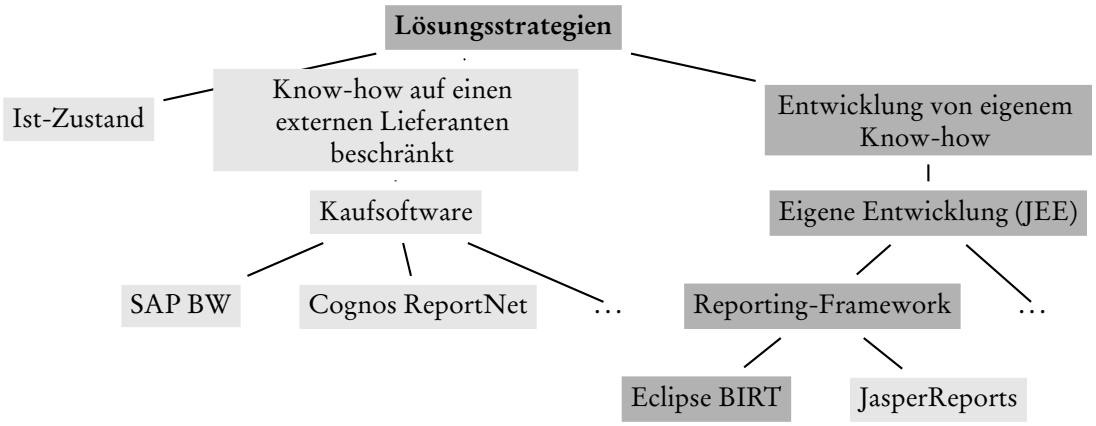
- Die Architektur der Anwendung – die Reports werden ausschließlich auf einem Desktop-System generiert – entspricht nicht den Anforderungen.
- Eine Migration auf das Nachfolger-System Cognos ReportNet wäre möglich.

4.2. Auswahl einer Strategie für die Reporting-Anwendung

Der am Projekt-Anfang erstellte Entscheidungsbaum für die möglichen Reporting-Anwendungen (s. *Abbildung 4.1*) beinhaltete folgende Lösungen, die jeweils unterschiedliche Strategie-Entwicklungen implizieren:

1. Einsetzen einer Kaufsoftware ohne Anpassungsmöglichkeiten
2. Eigene Entwicklung, basierend auf einem Reporting-Framework
3. Das Verbleiben im „Status-Quo“ – Beibehalten der existierenden Probleme – erwies sich schnell als inakzeptabel.

Abbildung 4.1.: Losungsstrategien für operatives Reporting: Entscheidungsbaum



In der *Abbildung 4.1* ist der verfolgte Lösungspfad als dunkelgrau markiert: Die Web-Applikation soll auf JEE-Architektur basieren, und die Reporting-Funktionen werden mit dem BIRT-Framework realisiert.

Im Folgenden werden die einzelnen Entscheidungen im Bereich Kaufsoftware kurz begründet.

DWH-Lösung: SAP BW Eine Analyse der vorhandenen Reporting-Anforderungen (4.1.2) und die Anwendung der „Kriterienliste für Auswahl einer Reporting-Lösung“ (3.2) deuten an, dass SAP BW (3.3.1) den Informationsbedarf im IPS-T-Umfeld nicht abdecken kann. In vielen

dieser Anforderungen geht es um tagesaktuelle Auswertungen, was eine DWH-Lösung per se nicht anbieten kann.

Cognos ReportNet ReportNet ist ein Web-basiertes Programm zur Erstellung und Verwaltung von Standard- und Ad-hoc-Reports der Firma Cognos (s. Cognos (2007)).

Ein möglicher Einsatz von Cognos ReportNet im IPS-T-Umfeld bringt folgende Nachteile mit sich:

1. Die Benutzer-bezogene Lizenzpolitik (auch für das Generieren von Reports) sowie sehr hohe Lizenz-Kosten (s. Kosten-Rechnungen). Die Anforderungen (s. *Anlage A*) können somit nicht erfüllt werden.
2. Eine geringe Akzeptanz bei den Reporting-Nutzern.
3. Schaffung einer Know-how-Abhängigkeit vom externen Lieferanten in einem strategisch wichtigem Umfeld (s. Eckerson (2002)).

Was die Realisierung einer Eigenen Entwicklung anbelangt, so wurde eine gemeinsame technische Basis in Form eines Reporting-Frameworks angestrebt.

Open-Source Reporting Frameworks Der Einsatz eines Open-Source Reporting-Frameworks innerhalb einer JEE-Musterlösung unterstützt die Entwicklung einer eigenen Strategie im Bereich des operativen Reporting und widerspricht nicht den Standardisierungszielen.

Eclipse BIRT vs. Jasper Reports Nach einer projektbegleichenden Marktrecherche wurden zwei Open-Source Projekte – Eclipse BIRT und Jasper Reports (Heffelfinger, 2006) – ausgewählt, untersucht und miteinander verglichen.

Aus einer Evaluierung (s. Schulz (2006)) dieser Frameworks geht hervor, dass Eclipse BIRT die ausgereiftere und damit bessere Wahl für einen produktiven Einsatz ist. Folgende Hauptgründe sprechen dafür:

1. BIRT weist vor allem in den Bereichen Integration, Verständlichkeit und Nutzbarkeit (Dokumentation, Einarbeitungsaufwand) sowie Erweiterbarkeit Vorteile im Vergleich zu Jasper Reports auf.
2. Die Entwickler-Community von BIRT ist dynamischer und größer. Die Entwicklung von BIRT wird von großen Firmen (Actuate, IBM, Pentaho, Scapa Technologies und Zend) unterstützt. Einige Firmen bieten auch kommerziellen Support für BIRT an.

3. Es ist zu erwarten, dass die Akzeptanz und Verbreitung von BIRT stark zunehmen wird, da es als Eclipse Projekt von der großen Community profitieren wird, welche Eclipse als IDE oder andere Eclipse Projekte nutzt.

4. Die Tools, die eine graphische Oberfläche zur Erstellung und Ausführung von Reports zur Verfügung stellen, werden bei BIRT direkt mitgeliefert.

5. BIRT weist ausgereifte Framework-Attribute auf. Aufbau und Design von BIRT basieren auf modernen Software-Konzepten, sind erweiterbar und gut dokumentiert.

Die vollständigen Ergebnisse der Evaluierung sind in der *Anlage B* aufgelistet.

4.3. Konzeptionelle Sicht

Abbildung 4.2 zeigt die konzeptionelle Sicht auf die Reporting-Applikation. Reports werden primär zum Ziel der Darstellung von Informationen aus einer oder mehreren *Datenquellen* verwendet. In dem konkreten Fall werden vor allem Daten aus der IPS-T-Datenbank (Legato-Datenbank) verwendet. Den Aufbau eines Reports und seiner Daten bestimmt seine *Beschreibung*. Sie wird mit einem *Report Creator* erstellt. Die *Report Engine* ist in der Lage, die Dateien zu interpretieren und Reports zu generieren: Sie greift auf die Datenbestände zu, bearbeitet die Ergebnisse und präsentiert sie entsprechend der vorliegenden Beschreibungen.

Der Reporting-Workflow sieht folgende *Rollen* im System vor:

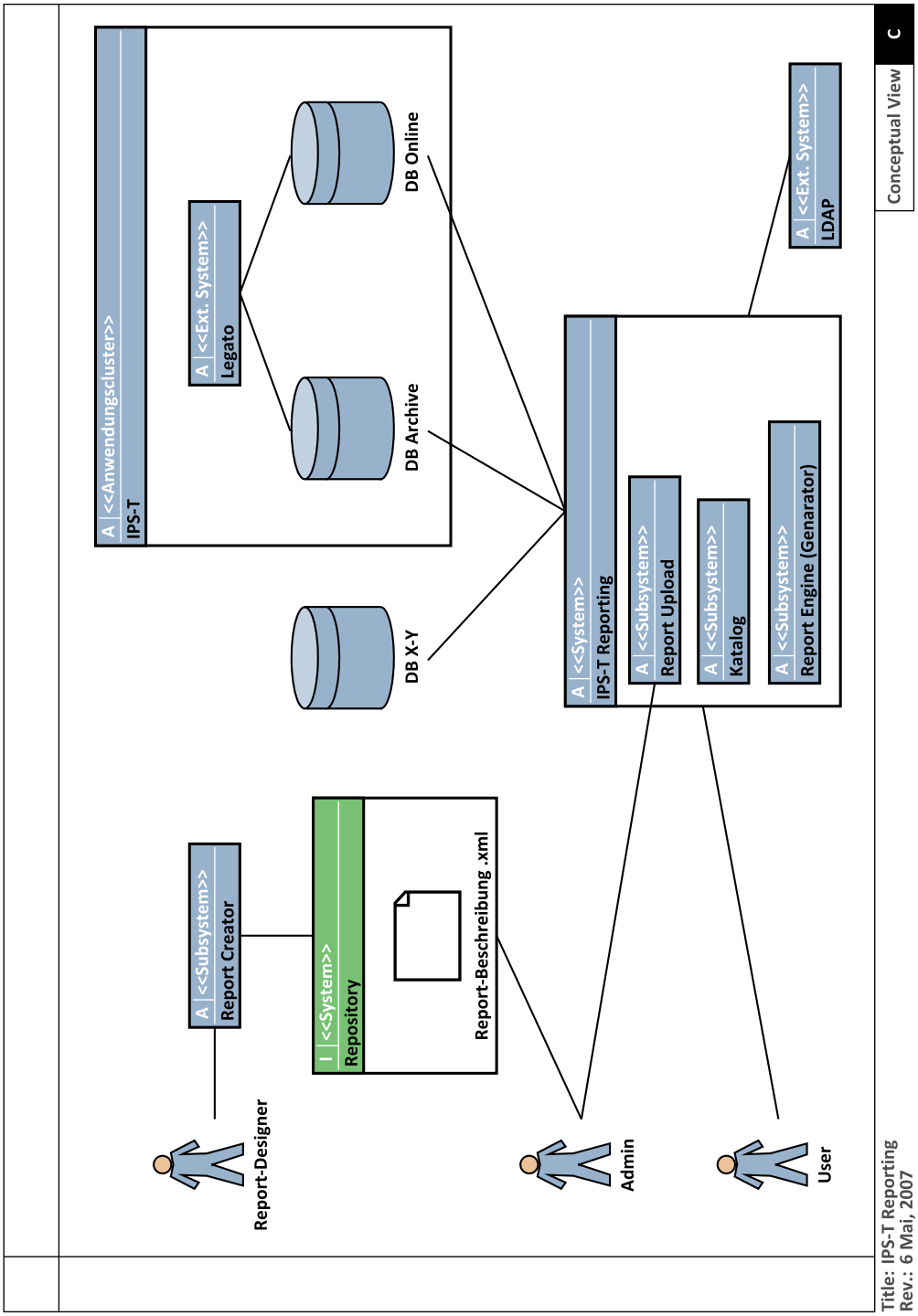
Die Reporting-User können Reports innerhalb einer *Web-Applikation* aufrufen. Die Web-Applikation verfügt über einen Katalog (eine Menü-Struktur), wo die Reports zu finden sind. Bei der Auswahl eines Reports können die Nutzer das gewünschte Ausgabe-Format auswählen. Bei sensiblen Reports werden sie zusätzlich aufgefordert, einen Passwort einzugeben.

Report-Designer entwerfen ihre Reports mit dem *Reporting-Editor*. In ihrer Arbeit gehen sie folgend vor:

- SQL-Abfragen definieren oder eine bereits vordefinierte Abfrage aus dem Abfragen-Katalog wiederverwenden.

- Das Layout eines Reports gestalten. Dabei werden Darstellungsformen, wie Tabellen, Listen, Texte, Diagramme etc. verwendet. Das Layout kann von einem Template entnommen werden. Die Element-Eigenschaften können mit Styles definiert werden.

Abbildung 4.2.: Reporting Konzeptionelle Sicht



- Die Ergebnisse einer Abfrage mit einer Darstellungsform verbinden. Dabei können unterschiedliche Funktionen angewendet werden, um die Daten zu gruppieren, sortieren, aggregieren oder zu filtern.

Auf diese Weise entstehen *Report-Beschreibungen*. Sie werden unter einem abgestimmten Speicherplatz abgelegt.

Der Administrator kann die von Report-Designer entworfenen Reports auf den Web-Server hochladen und innerhalb der Reporting-Web-Applikation einpflegen – parametrieren. Zusätzlich verwaltet er den *Reporting-Katalog* und kann die jeweiligen Einträge ändern oder erweitern. Der Administrator pflegt auch die *Sicherheits-Regeln* (Authentisierung und Autorisierung) für die Applikation und kann die Sichtbarkeit von Reports bestimmen.

4.4. Funktionale Sicht

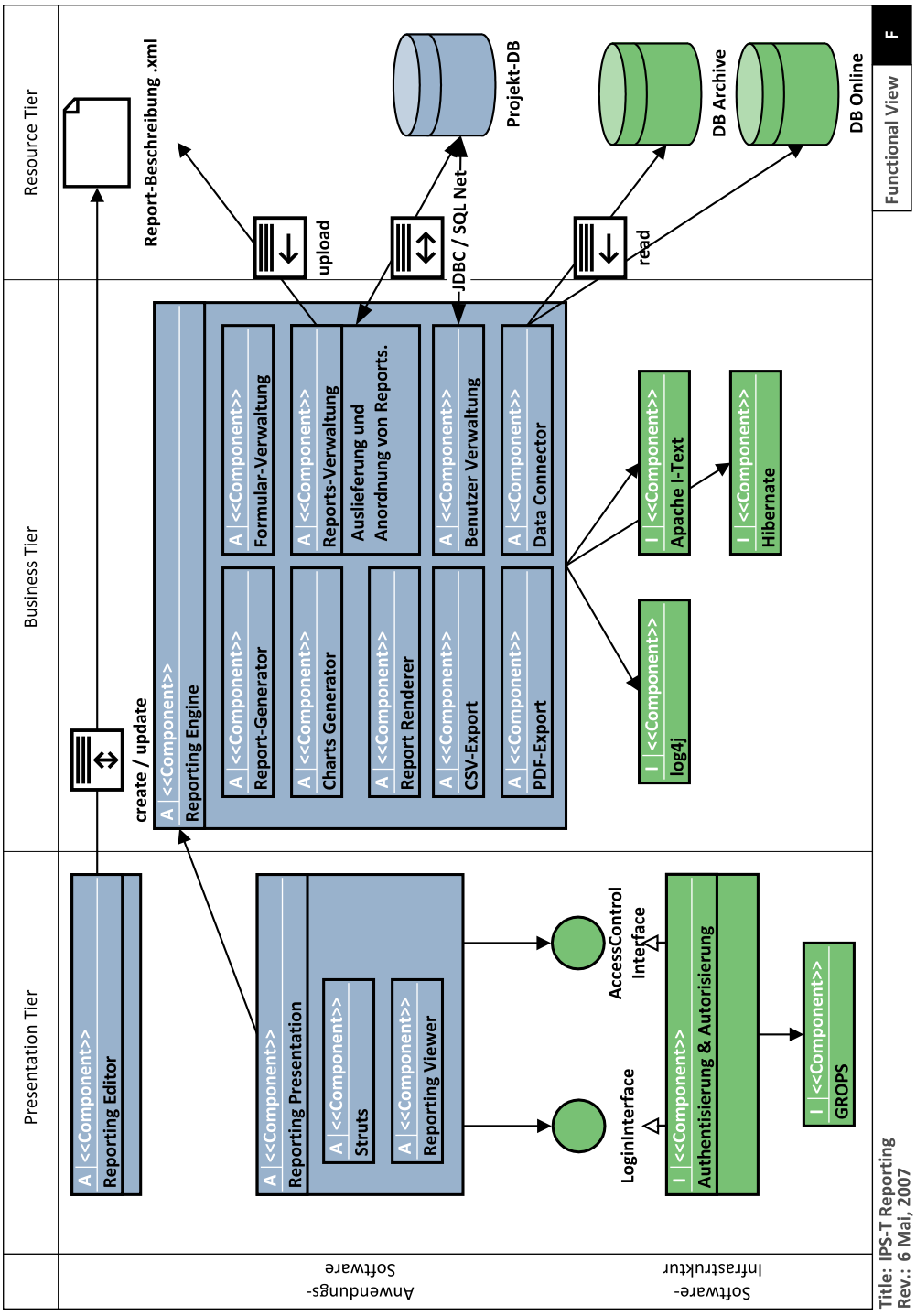
Abbildung 4.3 zeigt die funktionale Sicht auf das Reporting-System. Es besteht aus zwei Applikationstypen: Lokalen Clients und einer Web-Applikation. Mit einem Reporting-Editor werden die Reports erstellt bzw. geändert. Die Web-Applikation stellt bereits erstellte Reports den Nutzern zur Verfügung.

Die Web-Applikation ist nach ihrer Funktionalität in folgende Komponenten aufgeteilt:

- Reports präsentieren (Report Generator)
- Diagramme präsentieren (Chart Generator)
- Eingabe-Masken für die Report-Parameter verwalten
- Reports in unterschiedlichen Formaten ausgeben (z. B. CSV- und PDF-Export)
- Datenbankverbindungen verwalten
- Report-Kataloge verwalten
- Neue Reports hochladen (Upload)
- Neue Benutzer registrieren bzw. den existierenden Benutzern Rollen zuordnen (Benutzer-Verwaltung)

Auf die Reporting-Datenbank(en) kann die Applikation nur lesend zugreifen. Die benötigten Metainformationen (wie die Daten über Reports selbst, die Kataloge, die Daten über Benutzer) werden in einer gesonderten Projekt-Datenbank abgespeichert.

Abbildung 4.3.: Funktionale Sicht



Die Dienste der Benutzer-Authentisierung bietet ein zentraler LDAP-Server. Die Applikation verwendet öffentliche Schnittstellen des Services. Die Benutzer-Rollen werden von der Applikation selbst realisiert und verwaltet.

4.5. Entwicklungssicht

Die Entwicklungssicht beantwortet die Frage, wie eine Reporting-Applikation entwickelt wird.

Die Entwicklung der Reporting-Applikation kann in zwei große Bereiche aufgeteilt werden:

1. Die Entwicklung der Web-Applikation (s. *Abschnitt 4.5.1*)
2. Die Integration des BIRT-Frameworks in der Web-Applikation (s. *Abschnitt 6*) ggf. auch die Entwicklung fehlender bzw. zusätzlicher Reporting-Funktionen – Erweiterung von BIRT (s. *Abschnitt 5.6*)

In diesem Kapitel wird die erste Frage ausführlich beschrieben. Der zweiten Frage widmen sich die Kapitel 5 und 6 dieser Arbeit.

4.5.1. Entwicklung der Web-Applikation

Eine Web-Applikation kann nach dem im *Abschnitt 2.2.2* vorgestellten Schichten-Konzept entwickelt werden. Die Reporting Web-Applikation basiert auf der J2EE-Technologie.

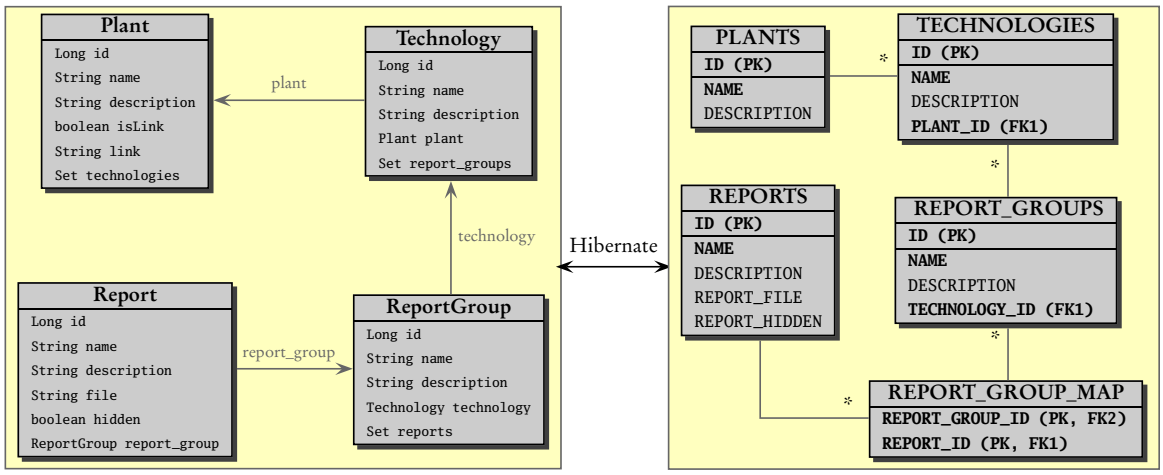
Datenmodell und Integrationsschicht
Die Geschäftslogik stellt ein logisches Modell der Anwendung dar, welches unabhängig von der Präsentation existiert. Sie enthält vor allem die Fachklassen der Anwendung und die Funktionalität zur Realisierung der Anwendungsfälle.

Der Reporting Web-Applikation liegt ein sehr einfaches Modell zugrunde: In der Applikation werden die Reports innerhalb einer Gruppe abgespeichert. Eine Werk-Technologie kann wiederum mehrere Report-Gruppen beinhalten. Die wesenlichen Business-Klassen der Applikation sind in der *Abbildung 4.4* gezeigt.

Die Daten des Models werden in einer relationalen Datenbank abgespeichert. Eine Abstraktionsschicht zwischen den Objekten der Applikation und der physikalischen Darstellung der Daten in der Datenbank wird von einem Objekt-Relationaler-Mapper (s. Fowler u. a. (2002)) hergestellt. Für diesen Zwecke verwendet die Reporting Web-Applikation Hibernate3[2], eine open-source ORM-Implementierung. Das Hibernate-Framework kapselt die üblichen Daten-Operationen (wie Select, Update, Delete, Insert) und verwaltet die Datenbank-Transaktionen. Eine exzellente Beschreibung des Frameworks findet man bei Bauer und King (2007).

[2] http://www.hibernate.org

Abbildung 4.4.: Reporting Web-Applikation: Business-Objekte und Datenmodell



Die Entscheidung, Hibernate3 einzusetzen, erscheint angesichts der Kompatibilität zum EJB3-Modell zukunftsorientiert und entspricht weitgehend den gängigen JEE-Standards.

Das Datenmodell wird zunächst als eine XML Mapping-Datei beschrieben (s. *Anlage C*). Aus diesen Dateien generiert man danach mit Hilfe der Hibernate Tools[3] die entsprechenden Java-Klassen und SQL-Anweisungen (s. *Abbildung 4.4*). Diese Vorgehensweise erhöht die Produktivität der Entwicklung und ermöglicht eine Datenbank-Kompatibilität, da die Statements automatisch von Hibernate bereits im jeweiligen SQL-Dialekt generiert werden.

Präsentationsschicht

Die Präsentationsschicht der Web-Applikation generiert auf die Anfrage eines Clients eine Beschreibung der Benutzeroberfläche und liefert sie dem Client (Browser) als HTML zurück. Diese Schicht wird meistens nach dem Model-View-Controller Pattern (MVC) realisiert (s. Fowler u. a. (2002) und Gamma u. a. (1995)). In der Java-Welt ist der Pattern auch unter dem Namen „Model 2“ genannt (Alur u. a., 2003) und wird u. a. mit JSP- JSF- und Servlets-Technologien realisiert.

Die Präsentationsschicht der Reporting-Web-Applikation wird mit dem Framework *Struts*[4] realisiert. Eine detaillierte Beschreibung des Frameworks würde den Rahmen der Arbeit sprengen, zusätzliche Informationen sind bei Husted u. a. (2003) zu finden.

Die Graphen-Darstellung der Reporting Web-Applikation und der Verbindungen zwischen den Struts-Komponenten zeigt die *Anlage D*.

[3] http://www.tools.hibernate.org
[4] http://www.jakarta.apache.org/struts

Logging

Das Erzeugen von Trace-Dateien dient dem Management und der Fehleranalyse sowohl in der Entwicklung als auch im produktiven Einsatz einer Applikation. In der Reporting Web-Applikation werden Logging-API von Log4j[5] eingesetzt.

Sicherheit

Durch den Authentisierungs- und Autorisierungsmechanismus der Web-Applikation können einzelne Seiten entsprechend ihrer URL geschützt werden.

Die URL einer geschützten Seite kann nur von bestimmten Benutzer-Gruppen aufgerufen werden. Die Zuordnungen „Benutzer – Rolle – Seiten“ werden vom Administrator der Reporting-Applikation vorgenommen und in der Datenbank abgespeichert. Für diese Zwecke stellt die Reporting Web-Applikation eine formularbasierte Administrations-Konsole zur Verfügung.

Die Authentisierung gegen einen LDAP-Server wird über das BMW GROPS-Framework realisiert. Der Autorisierungs-Mechanismus basiert auf der Filter Mapping Technologie von Java-Servlets und ist bereits vom BMW_XSAPPS-Framework realisiert. Diese Komponenten müssen lediglich in der Konfiguration der Web-Applikation (Web.xml) definiert werden.

Packages

Die Code-Struktur der Web-Applikation besteht aus mehreren Packages (Namensräume). Die *Tabelle 4.1* zeigt die einzelnen Packages der Web-Applikation und beschreibt sie kurz.

Tabelle 4.1.: Packages (Namensräume) der Reporting Web-Applikation

Schicht	Package	Funktionen
Model	`com.bmw.xsapps.reporting.objects`	Fachklassen der Anwendung
Integrationsschicht	`com.bmw.xsapps.reporting.daos`	Kapselt die CRUD-Operationen (Create, Read, Update, Delete)
View	`/html/content/reports/`	JSP-Seiten der Web-Applikation
	`com.bmw.xsapps.reporting.forms`	JavaBeans zur Aufnahme von Formulardaten
Controller	`com.bmw.xsapps.reporting.actions`	Klassen, die Zugriffe auf das Modell durchführen und dynamische Inhalte zur Darstellung an die zuständigen JSP-Seiten leiten
Querschnittkomponenten	`com.bmw.xsapps.reporting.util`	Hilfsklassen wie Datei-Upload, Streaming etc.

[5] http://www.jakarta.apache.org/log4j

4.6. Ausführungs- und Verteilungssicht

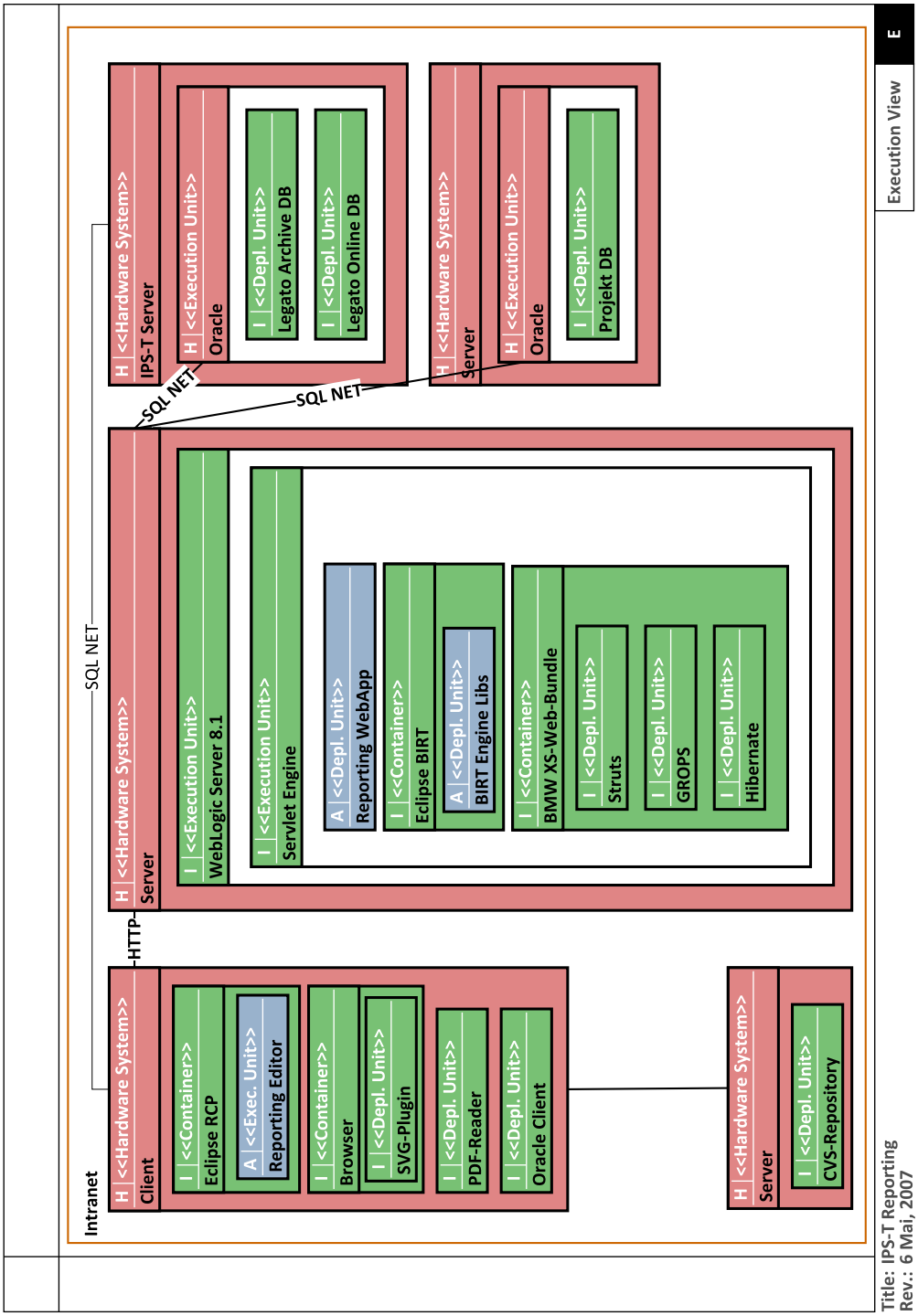
Abbildung 4.5 zeigt die Ausführungs- und Verteilungssicht auf das Reporting-System.

Client Eclipse BIRT Designer wird local auf einem Windows Rechner installiert. Zusätzlich werden ein Internet Browser, PDF- und SVG-Plugins benötigt. Über einen CVS-Client wird auf CVS-Reporting Repository zugegriffen.

Server Die Reporting-Web-Applikation und die BIRT Report Engine laufen unter einem Bea WebLogic 8.1 Application Server auf einem Linux SLES 9.

Infrastruktur Lesend greift die Applikation auf die *Reporting Datenbank* von IPS-T zu. Daten der Web-Applikation werden in einer *Projekt-Datenbank* abgespeichert. In beiden Fällen handelt es sich beim Datenbanksystem um Oracle 10g. Die Report-Dateien (Beschreibungen) werden auf einem Linux SLES 9 Server von einem CVS-Server verwaltet.

Abbildung 4.5.: Ausführungs- und Verteilungssicht



5. BIRT – ein Reporting Framework von Eclipse

Das Framework Eclipse BIRT wurde im Rahmen dieser Arbeit bei der Entwicklung der Reporting-Applikation verwendet und wird für den produktiven Einsatz im IPS-T Umfeld empfohlen.

Die Design-Konzepte und die Struktur von BIRT haben die im Kapitel 4 vorgestellte Software-Architektur für operatives Reporting stark beeinflusst. Daher werden sie in diesem Kapitel ausführlich analysiert.

Nach einer kurzen Beschreibung der Eclipse-Plattform wird zunächst die Funktionsweise von BIRT dargestellt. Danach werden die BIRT-Architektur und die Bestandteile des Frameworks detailliert beschrieben. Das Kapitel runden einige praxisorientierte Ausführungen in die Thematik BIRT-Scripting und BIRT-Erweiterungen ab.

5.1. Einführung

In Zusammenarbeit mit dem BI-Anbieter Actuate[1] rief die Eclipse Foundation im September 2004 ein neues Toplevel-Projekt ins Leben – *Business Intelligence and Reporting Tools (BIRT)*.

Mark Coggins, der Senior Vice President of Engineering der Actuate Corporation, schrieb über dieses Projekt:

> „We've leveraged our 10+ years of experience in the reporting and business intelligence space and put to work a significant number of full-time developers on the development of the platform. And while BIRT is a relatively young project that is only in its second major release, I think this investment is apparent in the rich and robust technology that the project provides. BIRT is an extensible, full-featured reporting platform that is ready for use in and integration with production applications."

Das BIRT-Projekt (Eclipse, 2007b) bietet eine Eclipse-basierte Entwicklungsumgebung und ein Java-Framework zum Erstellen und Generieren von komplexen, flexiblen und hochintegrierbaren Berichten an. Diese werden mit BIRT-Tools in graphischer Form entworfen. BIRT-Reports sind in der Lage, Daten aus mehreren (durchaus heterogenen) Datenquellen zu holen, sind in unterschiedliche Software-Umgebungen integrierbar und können die Ergebnisse in vielen Formaten ausgeben.

Das Ziel von BIRT ist es, den Aufwand für die manuelle Programmierung von Reporting- und Analyse-Funktionen in eigenen Projekten zu minimieren, indem alle Kernfunktionen in einem Framework bereitgestellt werden.

BIRT als Ansammlung von Werkzeugen und BIRT als Java-Framework bietet eine reizvolle Grundlage für eine betriebliche Reporting-Plattform an.

In diesem Kapitel werden die Funktionsweise, die Eigenschaften und die Kernarchitektur von BIRT sowie Integrationsmöglichkeiten des BIRT-Frameworks in eigenen Entwicklungen diskutiert.

5.2. Eclipse-Plattform

Bis zur Version 2.1 war Eclipse als eine Plattform für Softwarewerkzeuge konzipiert. Seit der Version 3.0 wurde sie neu organisiert und bietet nunmehr in ihrem Kern eine auf OSGi basierte Ablaufumgebung für Java-Module (so genannte Bundles und Services) an. Diese grund-

[1] http://www.actuate.com

legende Eclipse-Funktionalität wird als Rich Client Platform (RCP) bezeichnet. Alle Eclipse-Werkzeuge sind spezifische Rich-Client-Anwendungen und benutzen RCP Basisdienste.

5.2.1. Was ist RCP?

Die RCP ist ein Plug-In-basiertes Anwendungs-Framework. Erweiterbarkeit und Integration von Plugins sind zentral, das Laden der Bundles kann zur Laufzeit geschehen. RPC stellt auch eine Vielzahl allgemeiner Basisfunktionalitäten bereit, wie Benutzeroberflächen, Hilfesystem und Update Manager. Alle Eclipse-Projekte basieren auf der RCP, unter anderem Java Development Tools (JDT), C/C++ Development Tools (CDT).

BIRT baut auf RCP und OSGi Plattformen auf. Im Folgenden werden diese Technologien kurz vorgestellt. Eine fundierte Beschreibung des Frameworks ist in (McAffer und Lemieux, 2005), (Clayberg und Rubel, 2006) und (Daum, 2007) zu finden.

5.2.2. Architektur von Eclipse RCP

Eclipse RCP ist modular aufgebaut und beinhaltet fünf Hauptkomponenten (in der *Abbildung 5.1* sind sie im dunklen Kasten dargestellt):

UI Workbench bildet einen Rahmen für die RCP-Anwendung, welcher aus Perspektiven, Sichten (Views), Aktionen, Editoren, Dialogen und Wizards bestehen kann.

SWT (Standard Widget Toolkit), ein low-level Java-GUI-Framework.

JFace, eine SWT Abstraktionsebene, Anwendung des Model-View-Controller Patterns.

Platform Runtime, die Ablaufumgebung für jede RCP-Applikation. Sie kann über Erweiterungspunkte angepasst werden.

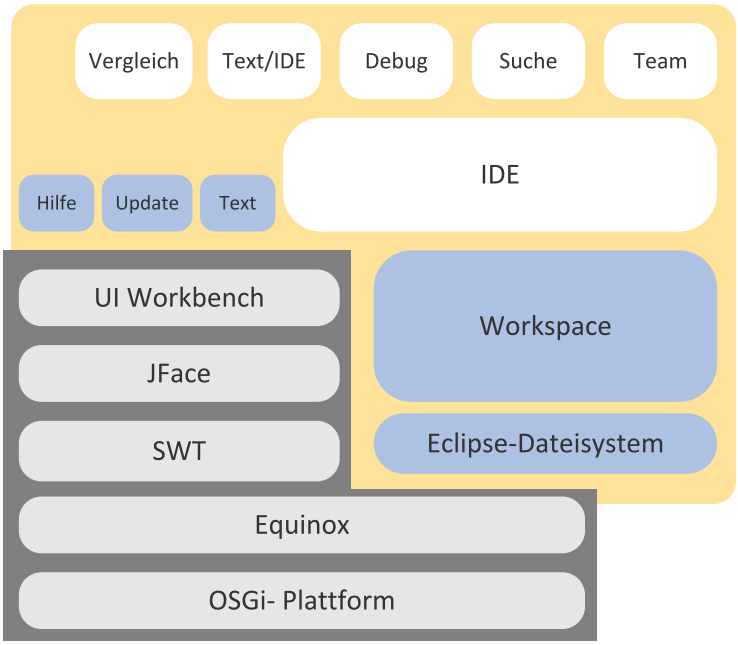
OSGi (Open Services Gateway Initiative), ein Framework, das von Eclipse für Plug-In-Verwaltung und Lifecycle-Management inklusive Hot-Plugging Funktionalität[2] benutzt wird[3].

Zusätzlich werden in der erweiterten RCP-Distribution auch Plug-Ins für Ressourcen-Management, Hilfesystem, Update Manager und Bausteine für die Textverarbeitung bereitgestellt (in der *Abbildung 5.1* sind das die hell-blauen Kästchen)

Metadateien
Alle RCP-Plug-Ins werden durch zwei Metadateien beschrieben: manifest.mf und plugin.xml.

[2] Das Laden /Aktivieren bzw. Entladen / Deaktivieren von Plug-Ins geschieht zur Laufzeit und benötigt nicht das Neustarten der Anwendung.
[3] Die Eclipse-Implementierung von OSGi heißt Equinox, siehe http://www.eclipse.org/equinox

Abbildung 5.1.: Architektur von Eclipse RCP



Eine Manifest-Datei legt Eigenschaften wie Name, Version, Einstiegspunkt, Lokalisierung und Abhängigkeiten fest.

Plugin.xml definiert eigene Erweiterungspunkte – „Extension Points“ und meldet Abhängigkeiten zu anderen Plug-Ins – „Extensions“.

5.2.3. RCP und BIRT: Das Zusammenspiel:

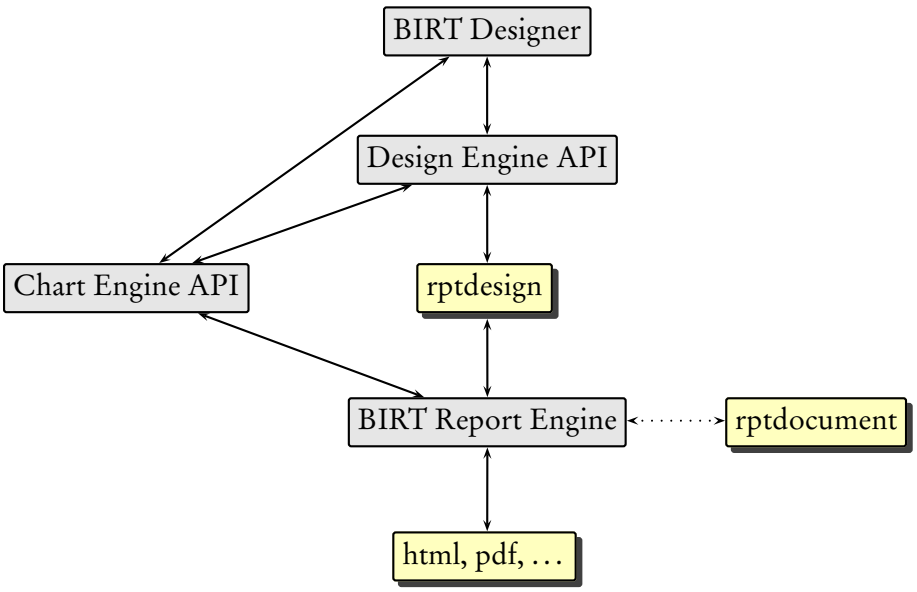
Die RCP-Architektur überträgt sich auf die darauf basierende Anwendung. Deren große Funktionsbasis, Plattformunabhängigkeit und flexible Integrationsmöglichkeiten erlauben die Konstruktion von multifunktionaler Anwendungssoftware. BIRT als ein Reporting-Framework passt optimal in eine Java-Entwicklungsumgebung. Funktionen für Debugging, Deployment oder Versionskontrolle können dadurch auch beim Entwerfen eines Reports ohne Mehraufwand angewendet werden.

5.3. Eclipse BIRT

5.3.1. Grundlegende Funktionsweise

Im Folgenden werden die üblichen Arbeitsprozesse von BIRT beschrieben. Das Zusammenspiel der einzelnen Schritte und jeweils entstehende Artefakte sind in der *Abbildung 5.2* dargestellt.

Abbildung 5.2.: Eclipse BIRT Workflow



Das BIRT-Herzstück ist der Report-Designer. Er besteht aus mehreren Eclipse-Plugins und bietet innerhalb einer Eclipse-Perspektive über unterschiedliche Paletten und Assistenten die Möglichkeit an, die Reports per „drag-and-drop“ zu entwerfen. Auf diese Weise entsteht eine Report-Beschreibung – Report-Design[4]. Sie wird standardmäßig als eine XML-Datei abgespeichert und kann auch im Nachhinein mit dem BIRT-Designer modifiziert werden. Über die BIRT Design Engine API kann eine Design-Beschreibung auch programmiertechnisch erstellt bzw. geändert werden.

Die BIRT Report Engine erstellt Reports aus den vom Benutzer entworfenen Report-Designs. Dieser Prozess geschieht in zwei Phasen: Generierung und Präsentation (Rendern).

Beim Generieren werden zuerst die nötigen Daten aus den definierten Datenquellen geholt. Die dadurch gewonnenen Informationen werden danach in einer Zwischendatei[5] abgespeichert.

Nach dem Rendern liefert die Engine entsprechend dem Entwurf eine Ausgabe der Ergebnisse als Html, Pdf oder Csv zurück.

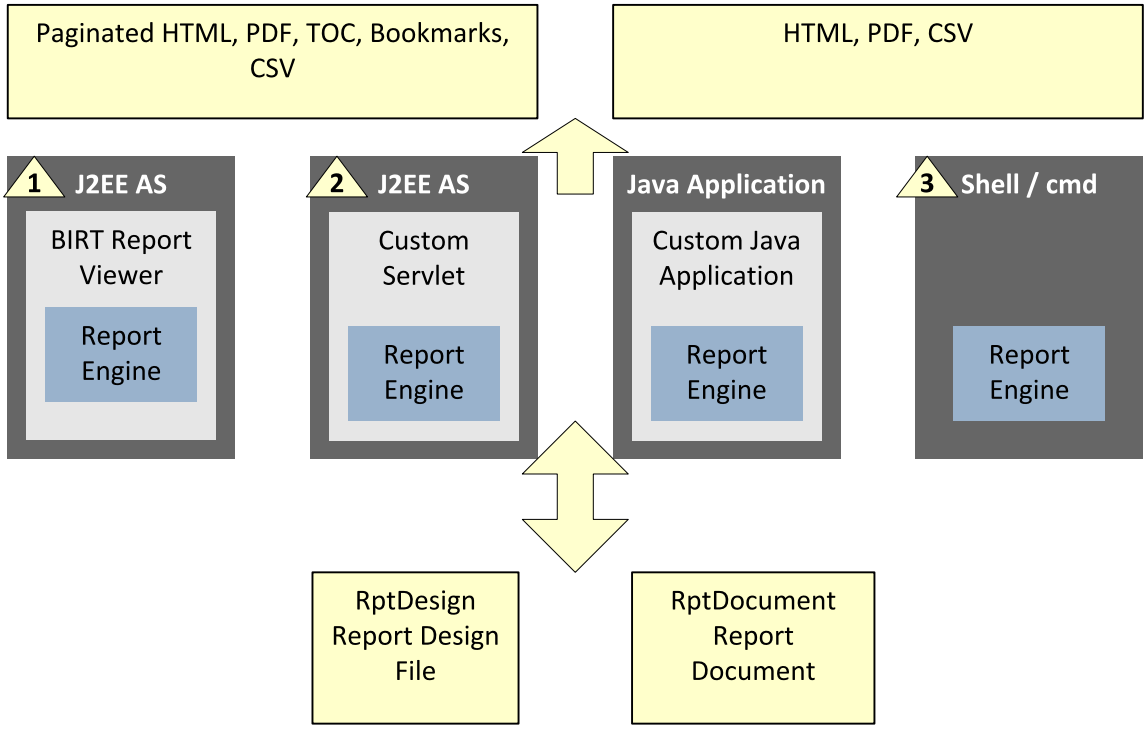
Die beschriebenen Phasen können abhängig von der Konfiguration (s. *Abschnitt 6.1.4* und *Abschnitt 6.3.6*) in einer Task oder getrennt – zeitlich entkoppelt – durchgeführt werden.

Wie in der *Abbildung 5.3* dargestellt, kann das Generieren der Reports auf unterschiedliche Art und Weise realisiert werden:

[4] .rptdesign
[5] .rptdocument

- über Report-Designer in einem Vorschau-Fenster. Gut geeignet für Testen der Report-Entwürfe und für Ad-hoc Reporting (s. Peh u. a. (2006)).
- eingebettet in einem Servlet bzw. einer JSP-Seite. Der Aufruf der Reports erfolgt über entsprechende Servlets des BIRT Viewers (s. *Abschnitt 6.1*) oder direkt über Report Engine Klassen (s. *Abschnitt 6.3*). Beide Varianten kommen beim Web-Reporting zum Einsatz und werden in dieser Arbeit ausführlich diskutiert (s. *Abschnitt 6*).
- direkt über die Report Engine innerhalb einer eigenständigen Java (RCP) Desktop-Applikation (s. *Abschnitt 6.3* und Weathersby (2006)).
- Aufruf der Report Engine über Kommandozeile. Auf diese Weise können Berichte automatisch und zeitgesteuert generiert werden.

Abbildung 5.3.: Integrationsmöglichkeiten von BIRT



5.3.2. Bestandteile eines BIRT-Reports

Ein Report in BIRT besteht aus folgenden vier Hauptkomponenten:

- den Daten,

- den Datentransformationen,
- der Geschäftslogik und
- der Darstellung.

Daten eines Reports
Die Daten eines Reports können aus Datenbanken, Web-Services oder Java-Objekten stammen. BIRT unterstützt JDBC zur Datenbankanbindung. Andere Datenquellen können über Java- bzw. JavaScript-Events eingebunden werden.

Datentransformationen
Datentransformationen sorgen in einem Report dafür, dass die Informationen den Wünschen des Endanwenders entsprechend sortiert, zusammengefasst, gefiltert oder gruppiert präsentiert werden. Diese Funktionen können entweder in der Datenbank realisiert werden oder aber von BIRT selbst, was oft von großem Interesse ist. Darüber hinaus stellt BIRT auch anspruchsvollere Operationen á la Excel, wie das Gruppieren von Summen, Mapping oder das Bilden von Prozenten, bereit.

Geschäftslogik
Um die vorliegenden Daten in nützliche Informationen zu konvertieren und dabei eigene, oft sehr raffinierte Geschäftslogik-Regeln zu beachten, kann sich die Scripting-Funktionalität von BIRT als behilflich erweisen.

Layout
Die Darstellung der Reports wird von BIRT in vielfältigen Ausprägungen unterstützt, dazu zählen z.B.:

Listen Die einfachste Art eines Reports ist die Darstellung von Informationen in Listen. Sie sind durch Gruppierungen und Aggregationen anpassbar.

Diagramme Numerische Daten können in einer grafischen Darstellung viel attraktiver präsentiert werden. BIRT unterstützt u.a. Kreis-, Linien- sowie Balkendiagramme.

Tabellen / Kreuztabellen (cross tables) Berichte können auch Tabellen oder Abfragen enthalten, die eine mehrdimensionale Sicht auf die Daten erlauben.

Texte / dynamische Texte / Bilder / Notizen BIRT erlaubt eine einfache Einbindung von weiteren Daten innerhalb eines Reports, wie beispielsweise von Notizen, Texten, weiteren Dokumenten oder Bildern. Die Daten können aus unterschiedlichen Quellen stammen: sowohl statisch aus einem Repository (Verzeichnis), als auch dynamisch aus der Datenbank.

Formatierungen Unter BIRT lassen sich Reports flexibel formatieren und mit Designvorlagen anpassen. Die Corporate Identity eines Unternehmens kann somit in einem Report leicht umgesetzt werden und über Libraries und Templates konsistent wiederverwendet werden.

Zusammengesetzte Reports (combined reports) Gebündelte Informationen zu einem Sachverhalt oder wichtige Nebeninformationen lassen sich mit BIRT in einem zusammengesetzten Report realisieren (z.B. kombinierte Reports aus Listen und Diagrammen).

Interaktive Reports BIRT unterstützt Reports, in welchen Endanwender Parameter verändern oder über Hyperlinks auf Sub-Reports zugreifen können.

5.4. BIRT-Framework. Architektur

BIRT ist ein gutes Beispiel einer komponentenorientierten Architektur. Erfolgreiches Wiederverwenden bzw. Erweitern eines Frameworks erfordert immer ein gutes Verständnis seiner Architektur. Konzeptionelle und strukturelle Fragen des BIRT-Frameworks werden auf den folgenden Seiten ausführlich beschrieben.

BIRT ist sehr eng in die Eclipse-Plattform und ihre Frameworks integriert. Entsprechend den Eclipse-Konzepten (s. *Abschnitt 5.2*) besteht BIRT aus einer Menge von Plug-Ins, die nach außen eine entsprechende Funktionalität bereitstellen. Zusätzlich verfügen BIRT-Plug-Ins über Kommunikationsschnittstellen zu anderen Frameworks. Über die BIRT-Erweiterungspunkte kann das Framework individuell angepasst werden.

Die Beziehungen zwischen einzelnen BIRT-Teilen lassen sich am besten als ein mehrschichtiges Gebilde visualisieren: Jede Schicht ist von der direkt darunterliegenden Schicht abhängig, weil sie auf ihren Funktionalitäten aufbaut. Die Abbildungen 5.4, 5.5 und 5.6 illustrieren unterschiedliche BIRT-Teile und deren Beziehungen untereinander.

In den nachfolgenden schematischen Darstellungen der BIRT-Architektur werden folgende Meta-Bezeichnungen verwendet:

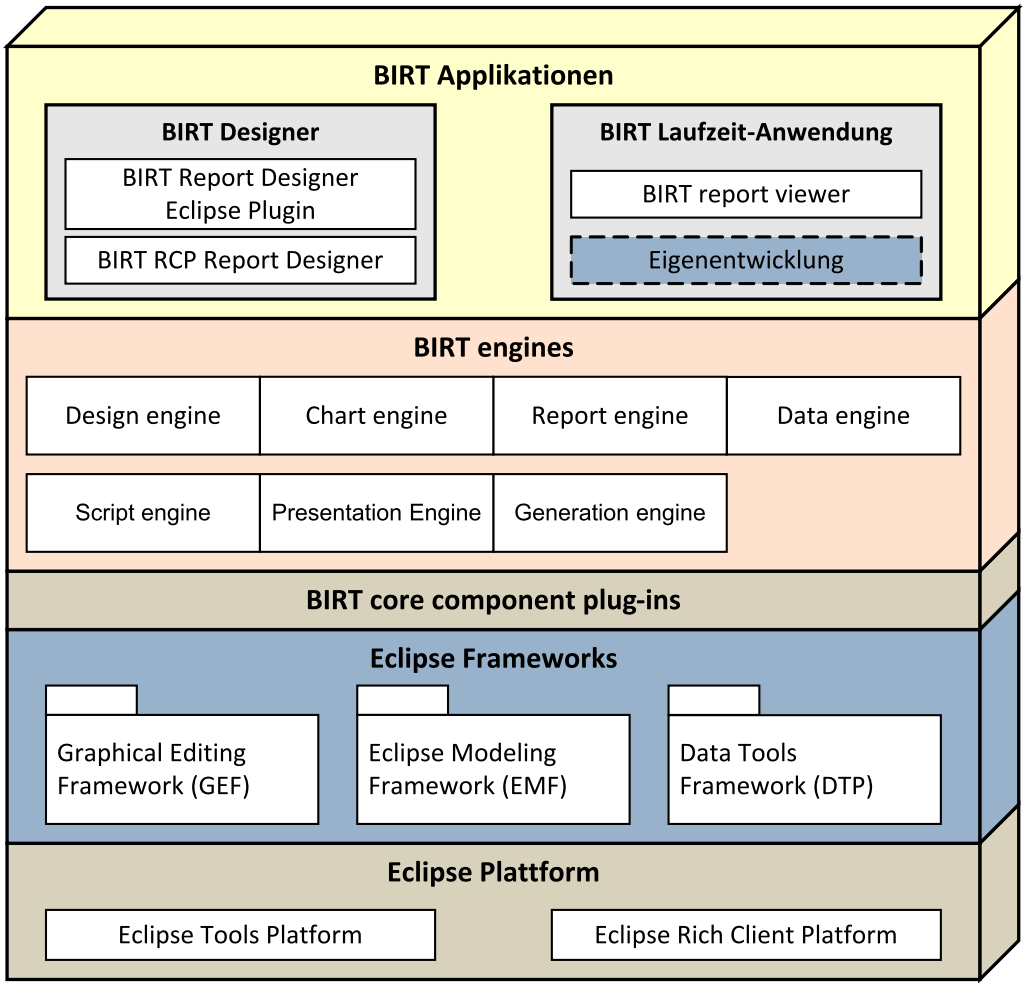
Data explorer	*massive Kästchen:*	eine Standard-Komponente bzw. Applikation von BIRT
Report item extensions	*gestrichelte Kästchen:*	eine eigene Entwicklung (Applikation oder Komponente, welche BIRT-APIs verwendet oder erweitert)
Report item extensions	*gestrichelte Kästchen mit blauem Hintergrund:*	ein Erweiterungspunkt (s. Kapitel...) von BIRT
Report previewer BIRT report viewer	*verschachtelte gepunktete Kästchen:*	eine Standard-Komponente von BIRT, die von anderen BIRT-Komponenten verwendet wird

5.4.1. BIRT Applikationen

Es gibt drei BIRT Applikationen:

- BIRT Report Designer,
- BIRT RCP Report Designer und
- BIRT Report Viewer.

Abbildung 5.4.: BIRT Komponenten und Plug-Ins (nach Weathersby u. a. (2006))



BIRT Report Designer (RD) und BIRT RCP Report Designer (RCP RD)

Beide Applikationen sind graphische Werkzeuge zum Erstellen bzw. Entwerfen von Reports. Sie verwenden die Report Design Engine, um die Reports-Designs zu erstellen. Reports-Designs

sind XML-Dateien und basieren auf der Syntax des Report Object Models (ROM). Im ROM sind alle erlaubten Elemente von einem BIRT-Report definiert.

BIRT Report Designer ist ein Eclipse Plug-In und erlaubt das Erstellen von Reports innerhalb der Eclipse-Workbench. **BIRT RCP Report Designer** ist eine RCP-Applikation (s. *Abschnitt 5.2*). RCR RD ist eine vereinfachte Version von RD und weist folgende Unterschiede auf:

- RCP RD kann zur gleichen Zeit nur eine einzige Design-Datei geöffnet haben.
- RCP RD hat keinen integrierten Debugger.
- RCP RD unterstützt nicht Java Event Handler.

BIRT Designer Applikationen bieten auch eine graphische Unterstützung für Templates und Libraries (s. *Abschnitt 5.4.4*).

BIRT Report Viewer (BRV)

BRV ist ein Java Web-Servlet zum Ausführen (Generieren und Präsentieren) von Reports in einem Web-Container. Aus einer Report Design Datei erstellt der BRV eine Report Document Datei, aus welcher dann eine Ausgabe im Zielformat generiert wird.

BRV weist eine weitere nützliche Funktionalität auf: die Eingabemasken für benutzerdefinierte Report-Parameter werden automatisch generiert.

BRV ist mit vielen Web-Containern und Applikations-Servern integrierbar. Erfahrungsberichte zur BRV-Integration mit Apache Tomcat, IBM WebSphere, BEA WebLogic und JBoss sind unter Eclipse (2007a) zu finden.

Im *Abschnitt 6.2* wird die Integration von BRV unter Bea WebLogic ausführlich beschrieben.

5.4.2. BIRT Engines

Eine Engine ist eine Java-Klassen-Sammlung (APIs), die Funktionalität für eine festdefinierte Domäne liefert. So bietet z.B. eine Daten-Engine APIs für Daten-Zugriffe an.

Report Design Engine (RDE)

RDE beinhaltet APIs für die Validierung (gegen ROM-Schema) und Generierung von Report-Design-Dateien. Sie wird vom BIRT Report Designer und der Generation Engine verwendet, kann aber auch in einer eigenen Applikation zum Einsatz kommen. RDE wird auch als eigenständige Implementierung, unabhängig von Eclipse, angeboten und kann in eigene Java-Projekten integriert werden.[6]

[6] Denkbare Entwicklungen wären hier u.a. ein Web Designer oder ein vereinfachter Wizard-basierter Designer.

Abbildung 5.5.: BIRT Report Designer: Standardkomponente und Erweiterungspunkte (nach Weathersby u. a. (2006))

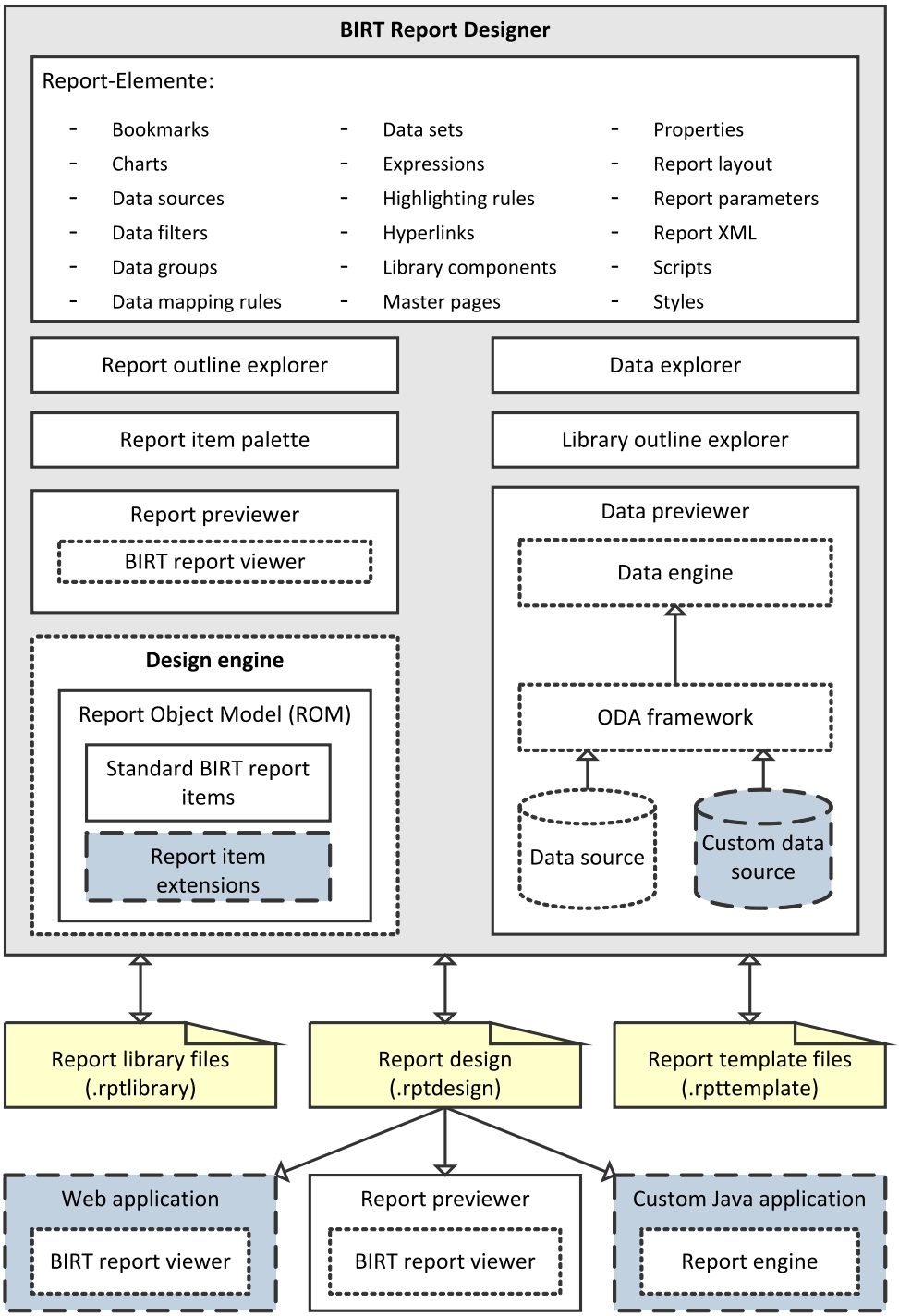
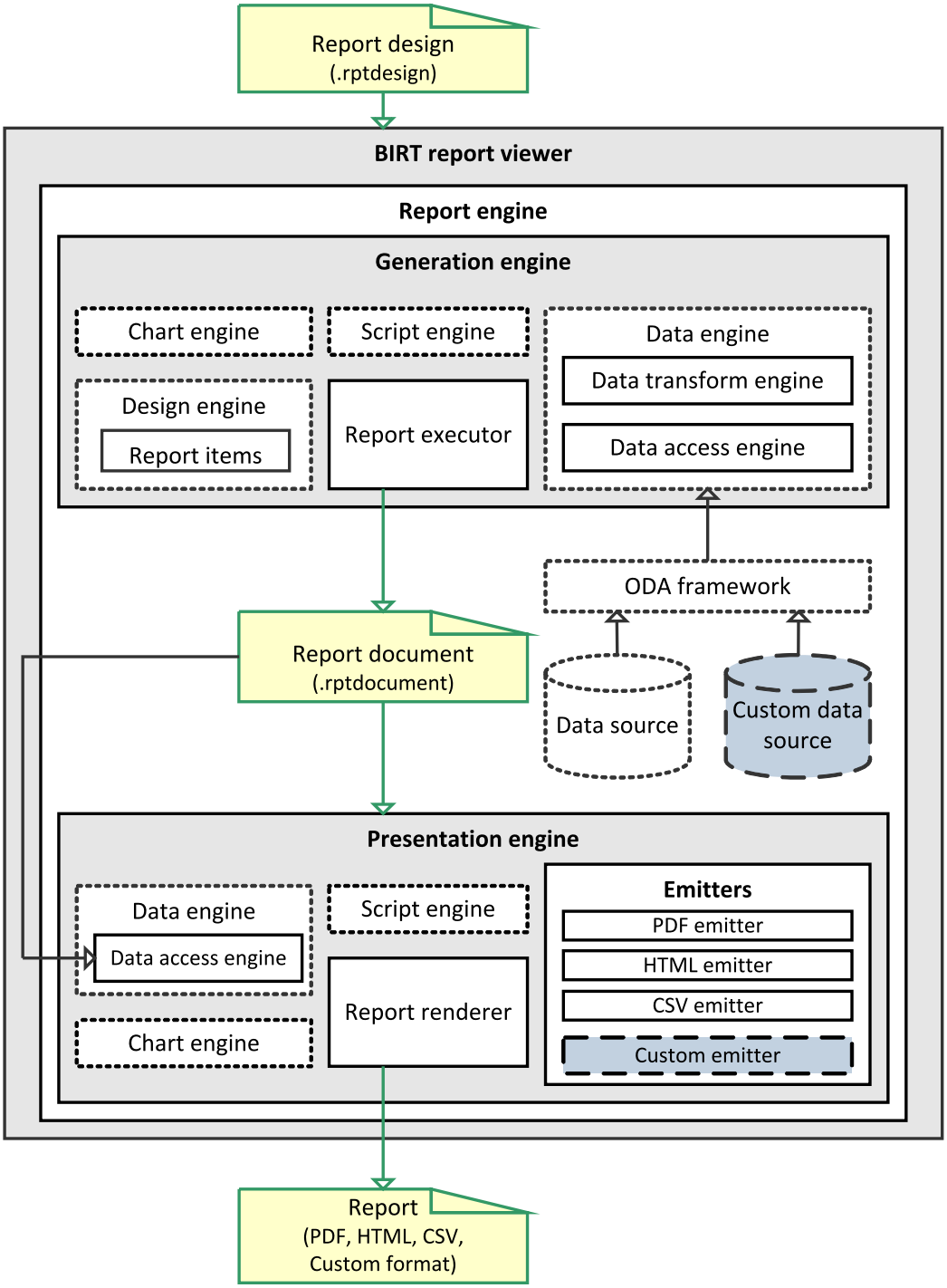


Abbildung 5.6.: BIRT Report Engine: Standardkomponente und Erweiterungspunkte (nach Weathersby u. a. (2006))



Report Engine (RE)

Eine RE besteht aus zwei Teilen: einer Generation Engine und einer Presentation Engine. Die erste wird beim Erstellen einer Report Document Datei eingesetzt, die Ausgabe in einem Report-Zielformat übernimmt dann die zweite.

Generation Engine (GE)
Eine GE beinhaltet folgende Funktionen:

- Report-Design-Datei lesen und interpretieren
- Daten aus den definierten Datenquellen anfordern, lesen und transformieren (dabei wird RDE verwendet)
- Ergebnisse in der Report-Dokument-Datei abspeichern

Presentation Engine (PE)
PE arbeitet mit Report-Dokument-Dateien und überführt sie in das Zielformat in folgenden Arbeitsschritten:

- Report-Dokument-Datei lesen und interpretieren
- Daten aus der Report-Dokument-Datei holen (dabei wird RDE verwendet).
- Einen zum Wunsch-Format (z. B. Pdf, Html, CSV) passenden Report Emitter lokalisieren und mit seiner Hilfe Report-Darstellung anhand der Beschreibung in der Report-Design-Datei generieren.

Damit auch die individuellen Report-Elemente bzw. Charts von der PE verstanden werden, muss PE von den Entwicklern mit entsprechender Funktionalität erweitert werden.

Chart Engine (CE)

Funktionen zum Erstellen und Generieren von Diagrammen (Charts) realisiert BIRT über ein eigenständige API-Sammlung – die Chart Engine. Sie beinhaltet auch APIs für die Herstellung der Bindungen zwischen Charts und Datenquellen.

Die CE ist nicht an BIRT gebunden und kann somit auch außerhalb der Eclipse- bzw. BIRT-Welt autonom weiterverwendet werden.

Eine Reporting-Applikation, wie z.B. BRV, liest eine Chart-Beschreibung aus der Design-Datei und verwendet die CE zum Generieren eines Charts.

Data Engine (DE)

Die DE hat die Aufgabe, Daten aus unterschiedlichen Datenquellen zu holen und zu transformieren. Sie besteht aus zwei Komponenten: Data Access (DEA) und Data Transform (DET).

DEA verwendet das ODA Framework, um auf die Datenquellen zuzugreifen und Daten-Abfragen zu starten.

DET ist für Operationen, wie Sortierung, Gruppierung, Aggregationen und Filter verantwortlich. Sie werden auf die Ergebnismenge, die DEA zurückliefert, angewendet.

Open Data Access Framework (ODA)

ODA ist ein Eclipse Framework für die gemeinsame und kollaborative Verwaltung von Datenquellen in einer RCP-Applikation. Es kann folgende Aufgaben übernehmen:

- Management der Datenbank-Treiber
- Laden der Treiber
- Öffnen der Connection
- Ausführen der Datenabfrage

Somit werden die nötigen Datenquellen (mit Treibern) einmal an einer Stelle definiert und stehen danach jedem RCP-Plug-In über einen Alias-Namen jederzeit zur Verfügung.

Die Art einer Datenquelle unter BIRT ist nicht auf relationale Datenbanken beschränkt. Da ODA über mächtige Erweiterungs-Konzepte verfügt, können neue, durchaus heterogene Datenquellen programmiertechnisch definiert werden: Entwickler müssen in diesem Fall die Custom Date Source mit einer gewünschten Logik (Connect-Methode) erweitern. (s. *Abschnitt 5.6*)

Unter BIRT sind in der Version 2.2 u.a. folgende Datenquellen möglich:

- Relationale Datenbanken
- Dateien (CSV- oder anders getrennte Daten)
- Web-Services
- Java-Objekte

5.4.3. BIRT Report-Elemente

Report Object Model (ROM)

ROM definiert Regeln für eine gültige Rptdesign-Datei und bietet damit ein Modell für BIRT Report Design-Beschreibung an. Die Spezifikation von ROM definiert eine Menge der XML-Tags zur Beschreibung der visuellen und nicht-visuellen Elemente (s. Weathersby u. a. (2006)) eines Reports:

Visuelle Elemente umfassen Tabellen, Listen, Labels etc.

Nicht-visuelle Elemente sind Report-Parameter, Datenquellen und Data Sets.

ROM Elemente beschreiben Seitenlayout, Style und Struktur von Report-Elementen sowie Datenquellen und Datenabfragen.

Die ROM-Spezifikation definiert all diese Elemente, ihre Eigenschaften und Relationen untereinander. Sie ist als XML Schema verfasst und beinhaltet eine formale Beschreibung der Inhalte, Struktur und Semantik für Report Designs.
Das ROM-Schema ist unter `http://www.eclipse.org/birt/2005/design` zu finden.

5.4.4. BIRT Dateien

Für die Beschreibung der Reports und Zusatzinformationen verwendet BIRT eine Reihe von XML-Dateien. BIRT Report Designer verwendet vier verschiedene Daten-Typen: Report Design Dateien, Report Document Dateien, Report Library Dateien, und Report Template Dateien.

Report Design Datei

Eine Report Design Datei ist eine XML Datei, welche die komplette Beschreibung eines BIRT-Reports beinhaltet. Sie besagt, wie Report-Struktur, Format, Datenquellen und Data Sets aussehen und welche Scripts ggf. verwendet werden sollen. Eine Report Design Datei wird vom BIRT Report Designer erstellt. Für das Generieren des Reports dient sie als Eingabe.

Report Design Dateien haben die Endung „`.rptdesign`“.

Report Document Datei

Eine Report Document Datei ist eine binäre Datei. Sie wird zum Zeitpunkt des Generierens eines Reports von Report Engine erstellt und dient als Transfer- bzw. Cache-Datei zwischen den Generation- und Presentation Engines. Eine Document Datei beinhaltet außer der Beschreibung des Report-Designs bereits die für den Report erforderlichen Daten.

Report Document Dateien haben die Endung „`.rptdesign`“.

Report Library Datei

Eine Report Library Dateien sind XML-Dateien, welche die wiederverwendbaren BIRT Report Komponenten beinhalten. Das können u. a. Bilder, Styles, visuelle Report-Elemente, Scripting-Code, Datenquellen und Data Sets sein. Eine Report Library Datei kann unter BIRT RD verwendet, erstellt und modifiziert werden. Die Komponenten erscheinen als eine Liste im Library Outline Explorer.

Report Library Dateien haben die Endung „`.rptlibrary`“.

Report Template Datei

Eine Report Template Datei ist eine XML-Datei, sie bietet eine Vorlage für Reports an und beinhaltet wiederverwendbare Report-Designs. Eine Report Template Datei kann u. a. Datenquelle(n) und Data Sets spezifizieren, das Layout oder Teile davon beschreiben, Kopf- bzw. Fußzeilen, Master-Pages, Gruppen oder Tabellen vorgeben.

Templates werden vom Entwickler für das Erstellen von neuen Reports verwendet, um den Arbeitsaufwand zu minimieren. Darüber hinaus ermöglichen sie vor allem das Einhalten der Corporate Design Regeln eines Unternehmens.

Report Template Dateien haben die Endung „`.rpttemplate`“.

5.5. BIRT Scripting

BIRT liefert mächtige Scripting-Fähigkeiten, die es den Entwicklern erlauben, unterschiedliche Aspekte beim Generieren von Reports programmiertechnisch zu beeinflussen. Damit kann die komplexe Funktionalität eines Reports (Geschäftslogik) mit Java oder JavaScript programmiert werden. BIRT RD bietet ein IDE zum Schreiben des Codes.

BIRT-Scripting basiert auf Mozilla Rhino (Rhino, 2007), einer Open-Source Implementierung von ECMAScript (JavaScript). Rhino beinhaltet zusätzlich JavaAdapter – Schnittstellen, die es ermöglichen, Java-Code in den Scripten zu verwenden. Die Synergie wird auch *Java-Scripting* genannt. Fundierte Informationen zum Thema ECMAScript und Java-Scripting sind bei Flanagan (2006) zu finden.

Unter BIRT kann Scripting in zwei Bereichen verwendet werden: zum Beschreiben von Ausdrücken (Expressions) und von Ereignissen (Events).

5.5.1. Expression Scripts

Expression Scripts sind kleine JavaScript Abschnitte (Snippets), die eine Berechnung durchführen und den Wert zurückgeben. Folgende Funktionen können mit den Expression Scripts durchgeführt werden:

- Aggregieren von Daten in Gruppen
- Spalten und deren Werte in einem Result Set berechnen
- Filtern von Daten
- Mappen von Werten (Zuordnung von Daten eines Wertebereichs auf die Daten eines anderen)
- Highlight von Elementen anhand von Bedingungen
- Dynamische Hyperlinks
- Dynamische Bilder
- Dynamische Texte
- Konvertierungen zwischen Datentypen

BIRT RD beinhaltet einen *Expression Builder*, mit welchem solche Ausdrücke gebaut werden. Venkatraman und Weathersby (2007) bieten ausführliche Informationen zum Expression-Scripting.

5.5.2. Event Scripts

BIRT Event Scripts sind JavaScript-Methoden, die an festen Stellen (Events) beim Erzeugen eines Reports ausgeführt werden. Das BIRT Scripting-Modell erlaubt, diese Methoden mit eigenem Code zu überschreiben. Dies ermöglicht eine flexibel anpassbare Ausführung von Reports.

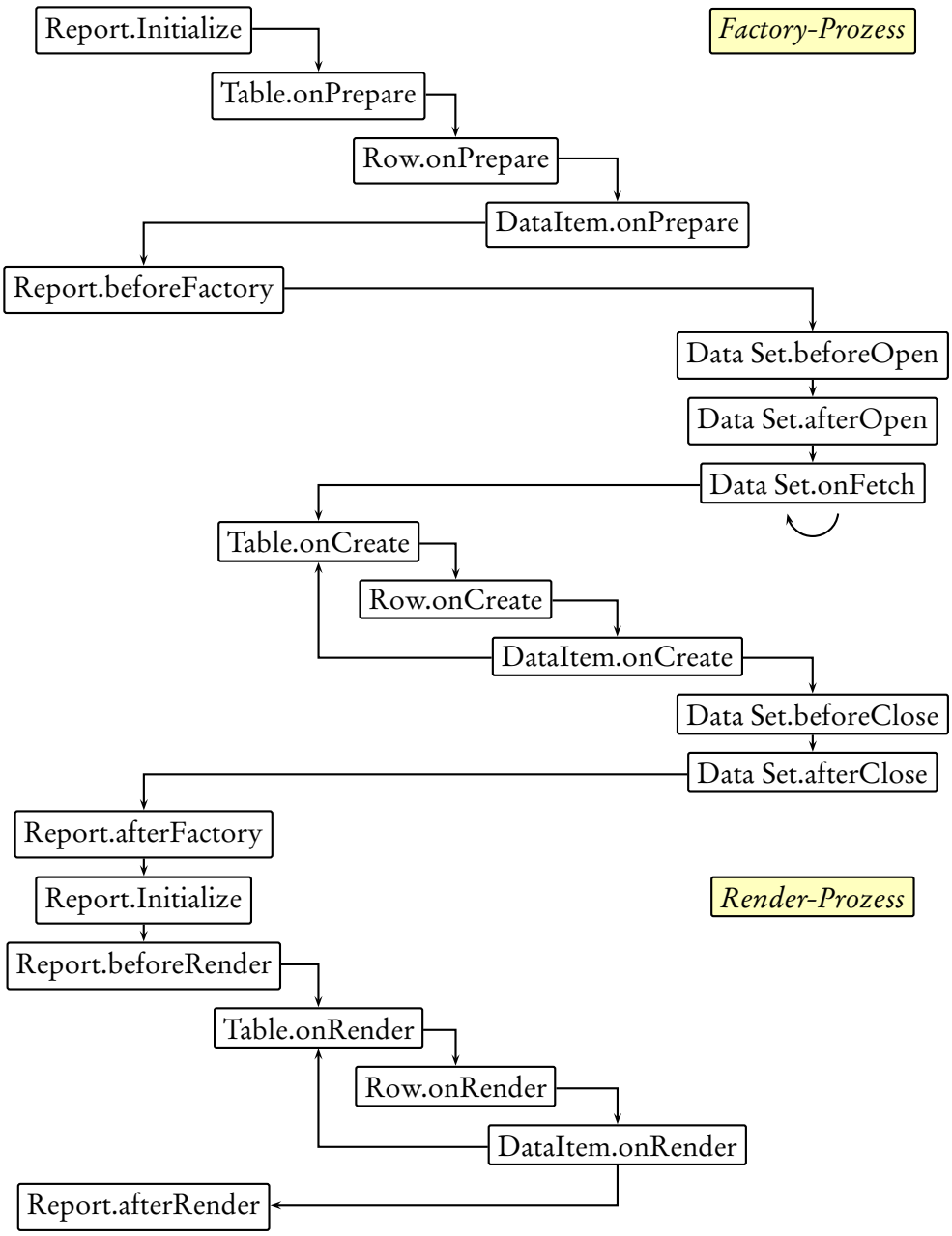
Ereignis-Modell von BIRT

Wie im *Abschnitt 5.3.1* schon beschrieben, werden die BIRT Reports in drei Phasen erstellt: der Initialisierungs-, der Generierungs- und der Präsentations-Phase. Diese Phasen können in zwei getrennten Prozessen ausgeführt werden[7]. Der *Factory-Prozess* beinhaltet jeweils die Initialisierungs- und Generierungs-Phase, der Render-Prozess die Präsentations-Phase. Der *Render-Prozess* kann sowohl unmittelbar nach dem Factory-Prozess als auch zeitlich entkoppelt erfolgen. Da die Scripting-Variablen prozessgebunden sind, ist die Reihenfolge und Verteilung der Event-Methoden wichtig (s. *Abbildung 5.7*).

Script-Events sind für folgende Objekte definiert: die Reports selbst, die Report-Elemente, die Datenquellen und Data Sets (s. *Tabelle 5.1*).

[7] Eine Ausnahme bildet der BIRT Report Designer: Im RD werden die Reports innerhalb eines einzigen Prozesses ausgeführt.

Abbildung 5.7.: BIRT Scripting Events Pipeline



BIRT Scripting Beispiele

Im Folgenden werden einige BIRT Scripting Beispiele ausgeführt. Detaillierte Informationen zum Scripting unter BIRT findet man bei Weathersby u. a. (2006).

Tabelle 5.1.: BIRT Scripting Events

Generation Phase			**Presentation Phase**	
Report Level	Data Source/Set	Report Element	Report Level	Report Element
Initialize beforeFactory afterFactory	beforeOpen afterOpen onFetch beforeClose afterClose	onPrepare onCreate	beforeRender afterRender	onRender

Globale Variablen

Zugriff auf die globale Variablen eines Reports kann über das Objekt *reportContext* erfolgen. Die darunter abgespeicherten Werte stehen allen Report-Elementen in beiden Ausführungsprozessen zur Verfügung.

Die Variablen werden innerhalb des *Initialize-Events* des Report-Objektes gesetzt:

```
reportContext.setPersistentGlobalVariable("testglobal", "test global string");
```

Zugriff auf die globalen Variablen kann im *Prepare-Events* eines Report-Elementes geschehen, z. B. eines Labels:

```
this.text = reportContext.getPersistentGlobalVariable('testglobal');
// Das Label wird den Wert "test global string" bekommen
```

Der Zugriff auf die *Session-Variablen* wäre auch über das reportContext Objekt möglich:

```
// attributeBean ist die nötige Session-Variable der Web-Applikation
myAttributeBean = reportContext.getHttpServletRequest().getAttribute('attributeBean')↘
    ⟶;
docName = myAttributeBean.getDocumentName();
this.text = docName;
```

Element-Eigenschaften

Die Eigenschaften von Labels, Texten, dynamischen Texten und Tabellen-Zellen werden über die Report-Element-Events geändert.

Eine *Label-Beschreibung* kann in der Methode für den onPrepare- oder onCreate-Event geändert werden:

```
this.text = "My New Label"
```

Der neue *Text* kann innerhalb des onPrepare-Events gesetzt werden:

```
this.content = "My New Text"
```

Ähnlich sieht es auch bei den dynamischen Texten aus:

```
this.valueExpr = "row['CITY']";
this.valueExpr = "'my row count: ' + (row[0] + 1)";
```

Der Unterschied zwischen den onPrepare- und onCreate-Events liegt darin, dass sich der erste auf die Report-Elemente und der zweite auf die einzelnen Element-Instanzen bezieht. Das folgende Beispiel verdeutlicht das:

```
// Event onPrepare, Element Row
// Alle Zeilen werden rot:
this.getStyle().backgroundColor = "red";

// Event onCreate, Element Row:
// Nur die Zeilen mit einem Wert > 100 werden blau
if (this.getRowData().getExpressionValue(3) > 100)
    this.getStyle().backgroundColor="blue";
```

Eigenschaften von Datenquellen und Data Set

Wenn es sich bei einer Data-Source um eine Datenbank handelt, möchte man oft die *Verbindungseigenschaften* wie *Passwort* oder *Connection-Strings* geheim halten. Den Zugriff auf diese sensiblen Werte kann man in einem externen Java-Objekt kapseln. Das Objekt wird in der beforeOpen-Methode von der Datenquelle initialisiert, die nötigen Werte werden ausgelesen und den Report-Eigenschaften zugewiesen:

```
currentPassword = this.getExtensionProperty("odaPassword");
DataSourceClass = new Packages.myExternalSecurity();
this.setExtensionProperty("odaPassword", DataSourceClass.getPassword());
```

Eine *Datenbank-Query* kann in der beforeOpen-Methode vom Data Set geändert bzw. gebaut werden:

```
this.queryText = "SELECT * FROM Customers where CustomerID IN (" + params["↘
    ⟶customersInClause"] +")";
```

5.6. BIRT Erweiterungen

Wie jeder Framework kann BIRT über die „Erweiterungspunkte“ (s. *Abschnitt 2.6.1*) angepasst und erweitert werden.

Unter BIRT sind die folgenden Komponenten erweiterungsfähig:

Report Items Definition von neuen Report Elementen oder Anpassungen von existierenden Elementen. Erweiterungspunkte sind die Klassen:
`org.eclipse.birt.report.model.reportItemModel`,
`org.eclipse.birt.core.ui.Tasks` und `taskWizards`.

BIRT-Funktionen Es können neue Aggregations-Funktionen implementiert werden. Den Erweiterungspunkt stellt die Klasse `org.eclipse.birt.data.Aggregation` bereit.

Emitters Schnittstelle zum Anbinden neuer Ausgabe-Formate. Standardmäßig werden mit BIRT HTML-, PDF- und CSV-Emitters mitgeliefert. Die entsprechenden Erweiterungspunkte befinden sich im Package `org.eclipse.birt.report.engine.emitters`.

Charts Definition von neuen Diagramm-Typen. Die entsprechenden Erweiterungspunkte sind in Packages `org.eclipse.birt.chart.engine` und `org.eclipse.birt.chart.ui` zu finden.

Data Sources Schnittstelle zum Anbinden neuer Datenquellen. Die Erweiterungpunkte werden vom Eclipse ODA Framework (s. *Abschnitt 5.4.2*) verwaltet. Rosenbaum (2007) bietet eine gute Einführung in diese Problematik.

6. Anwendung und Integration von BIRT

Die im Rahmen der Arbeit entwickelte Reporting-Applikation verwendet die aus dem Kapitel 5 bekannten APIs der BIRT Report Engine, um Auswertungen aus Report-Design-Beschreibungen zu generieren.

Dieses Kapitel vervollständigt die im Kapitel 4 beschriebene Entwicklungssicht der Reporting-Architektur. Es wird hinterfragt, welche Möglichkeiten für eine Integration von BIRT existieren und wie diese realisiert werden können. Dabei wird vor allem auf die zwei grundlegenden Vorgehensweisen für eine Integration von BIRT eingegangen:

1. Die Anbindung der Funktionen vom BIRT Report Viewer in einer Anwendung
2. Die Implementierung einer Reporting-Applikation mit den BIRT Report Engine APIs

Zu jedem dieser Szenarien wird jeweils eine Beschreibung der theoretischen Hintergründe und der praktischen Erfahrungen vorgelegt.

6.1. BIRT Report Viewer

Eine der Möglichkeiten, Reports im Web verfügbar zu machen, ist mit dem Einsatz von BIRT Report Viewer (BRV) gegeben. BRV ist eine AJAX-basierte JSP/Servlet Applikation mit Schnittstellen zu wichtigen BIRT-Funktionen. Sie ist damit in der Lage, Reports zu generieren und anzuzeigen.

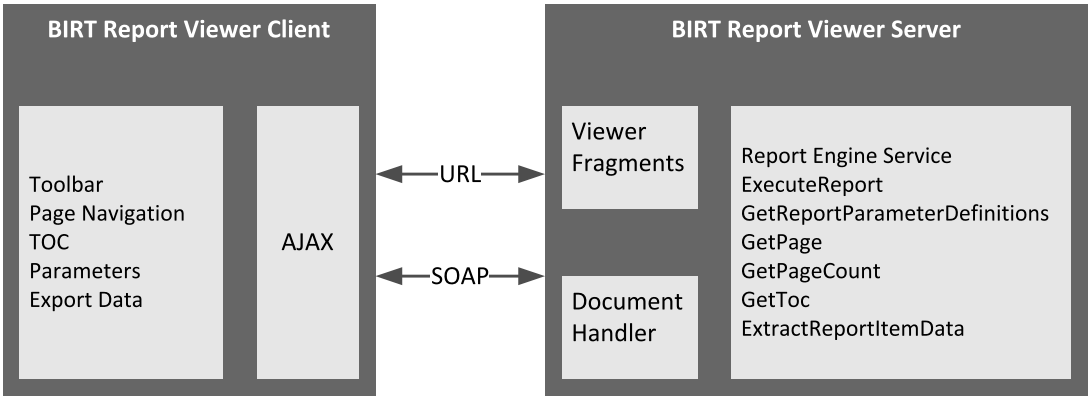
Eine Beschreibung von BRV sowie seiner Anordnung im BIRT-Umfeld befindet sich im *Abschnitt 5.4.1*. In diesem Kapitel werden konkrete Anhaltspunkte für eine Integration des BRVs mit Weblogic, dem Application Server von Bea, angeboten.[1] Eine Beschreibung für eine BIRT-Integration unter JBoss findet man bei Weathersby (2007).

6.1.1. Aufbau von BRV

Funktionsweise, Klassen

Die Struktur von BRV wird in der *Abbildung 6.1* dargestellt.

Abbildung 6.1.: Aufbau von BRV



Basisdienste von BRV wurden als Service-basierte Architektur realisiert. Die Kommunikation zwischen Client und Server-Diensten geschieht über SOA Protokoll und ist asynchron (AJAX).

Die Präsentationskomponente von BRV heißt **BIRT Report Viewer Client**. Sie bietet eine Web-Oberfläche mit folgenden Funktionen:

- Toolbar
- Seiten-Navigation

[1] Im Rahmen dieser Arbeit kamen BIRT in der Version 2.2M5 und Bea Weblogic 8.1 Service Pack 4 zum Einsatz.

- TOC
- Parameter-Eingabe
- Daten-Export

BIRT Report Viewer Server implementiert u.a. folgende Dienste:

- *Report Engine Service* zum Initialisieren und Starten der BIRT RE (Klasse `ReportEngineService.java`)
- *ExecuteReport* zum Ausführen der Reports (u.a. Klasse `DocumentProcessor.java`)
- *GetReportParameterDefinitions* zum Auslesen der Parameter-Definitionen aus der Design-Beschreibung (Klasse `ParameterAccessor.java`)
- *GetPage*
- *GetPageCount*
- *GetToc*
- *ExtractReportItemData*

Anpassung von BRV

Da BRV Open-Source ist, können seine Funktionen angepasst, geändert oder erweitert werden. Die Änderungen können JSP-Fragmenten, JavaScript-Dateien und CSS-Styles betreffen.

6.1.2. BRV Servlets

Konzeptionell besitzt BRV zwei Arbeitsmodi, welche über Servlet-Mapping `run` und `frameset` aufrufbar sind.

frameset

Über Frameset-Mapping kann eine AJAX-basierte Schnittstelle zum Anzeigen von BIRT-Reports aufgefasst werden. Über eine Buttonliste werden Kommandos, wie Eingabe eines Parameters, Einblenden des Inhaltsverzeichnisses (TOC), Export nach Excel, Ausdrucken als PDF sowie Seiten-Navigation innerhalb eines Reports ausgeführt.

Die Schnittstellen bzw. GUI-Teile für die entsprechenden Funktionalitäten sind bereits von BRV realisiert.

run

Beim Verwenden des Run-Servlets werden die Reports in einem bestimmten Format – HTML oder PDF – zurückgegeben. Parameter-Eingabemasken müssen in diesem Fall vom Benutzer implementiert werden, HTML-Ergebnisse werden als eine einzige Seite dargestellt.

6.1.3. URL-Aufbau und Parameter

Die URL zum Aufrufen von BRV ist wie folgt aufgebaut:

```
http://sitePrefix/name-der-brv-app/frameset?Parameter-Liste
http://sitePrefix/name-der-brv-app/run?Parameter1=Wert1&Parameter2=Wert2
```

Beide Servlets unterstützen URL-Parameter (Namen und zugehörige Werte). Darüber hinaus können unterschiedliche Informationen bzw. Ausführungsaspekte eines Reports bestimmt werden (s. *Tabelle 6.1*).

Tabelle 6.1.: URL-Parameter für BRV nach (Weathersby u. a., 2006)

Parameter	Wert	Ist erforderlich?	Standardwert
`__report`	Pfad zu Report-Design-Datei (`.rptdesign`)	ja (wenn `__document` angegeben , dann nicht)	kein
`__document`	Pfad zu Report-Dokument-Datei (`.rptdocument`)	ja (wenn `__report` angegeben, dann nicht)	kein
`__format`	html oder pdf	optional (wird nur mit `run` verwendet)	html
`__locale`	Locale-Code	optional	JVM Locale
`__page`	Seitennummer	optional (wird nur mit `frameset` verwendet)	1
`__svg`	`true` oder `false`	optional (wird nur mit `run` verwendet)	`false`
Parametername	Benutzerdefinierte Werte	wie im Report-Design spezifiziert	

Der Parameter `__report` bzw. `__document` sind obligatorisch, andere Parameter sind optional.

__report

Der `__report` Parameter gibt den Namen der auszuführenden Report-Design-Datei an. Ein Test-Report im Verzeichnis REPORTS wird beispielsweise über folgenden URL-String im Browser aufgerufen:

```
http://localhost:8080/report-app/frameset?\_\_report= c:\_da\workspace\REPORTS\Test.↘
    ⟶rptdesign
```

__document

Der __document Parameter spezifiziert die Report-Dokument-Datei. Wie im *Abschnitt 5.4.2* beschrieben, ein Report wird in zwei Phasen kreiert: Generation und Presentation. Eine Zwischendatei – Report-Dokument – wird vom BRV immer automatisch erstellt.

Wenn der Parameter __document verwendet wird, bildet der BRV ausnahmsweise keine Report-Dokument-Datei neu, sondern benutzt eine unter dem spezifizierten Pfad bereits vorhandene Datei wieder. In diesem Fall wird die BIRT Generation-Phase nicht ausgeführt. Die Vorgehensweise erweist sich als sehr nützlich, wenn die Datenbankabfragen für einen Report umfangreich sind und Datenänderungen eher selten geschehen.

Der Pfad zu einem Report

Der Pfad zu einem Report-Design bzw. Report-Dokument kann sowohl absolut als auch relativ sein. Relative Angaben beziehen sich auf die Web-App Konfiguration[2] (s. *Abschnitt 6.1.4*).

__format

Wird ein Report über den Run-Servlet aufgerufen, so gibt der __format Parameter eine Ausgabe-Beschreibung an.

__locale

Unabhängig von der Web-App Konfiguration (s. *Abschnitt 6.1.4*) kann die Sprache eines Reports[3] über Locale-Wert[4] bestimmt werden.

__svg

Der Parameter wird beim Run-Servlet verwendet und schaltet eine SVG-Ausgabe für Diagramme an.

Report Parameter

sind im Report-Design spezifiziert und werden an dieser Stelle (Request) über Ihre Namen mit angegebenen Werten initialisiert. Ein Report Test.rptdesign mit dem String Parameter MyParameter kann beispielsweise folgend aufgerufen werden:

```
http://localhost:8080/report-app/frameset?\_\_report= c:\_da\workspace\REPORTS\Test.↘
    ⟶rptdesign&MyParameter=Hallo Welt!
```

[2] Web.xml
[3] Entsprechend der ROM-Spezifikation (s.*Abschnitt 5.4.3*) sind die BIRT-Elemente internationalisierbar.
[4] Die Werte sind dem ISO-Standard zu entnehmen. Für Deutsch wird beispielsweise die Zeichenkette „de_DE“ und für Englisch „en_US“ verwendet.

6.1.4. BRV Kontext-Parameter in Web.xml

Um diverse Arbeitsordner für Report-Design-Dateien, Bilder oder Log-Dateien an einer zentralen Stelle fest zu halten, verwendet BRV Kontext-Parameter einer Web-Applikation. Diese werden in der Datei `Web.xml` definiert. Abhängig davon, ob die Applikation als War-Archiv oder einfach als Ordner-Struktur deployed wird, können einige Werte erforderlich oder nicht erforderlich sein.

BIRT_VIEWER_IMAGE_DIR
Ist ein absoluter Pfad zum Ordner, wo erstellte Bilder und Diagramme abgespeichert werden können.
Notwendig für War-Deployment.
Optional für Nicht-War-Deployment.

BIRT_VIEWER_LOG_DIR
Ist ein absoluter Pfad zum Ordner, wo Log-Dateien mit BIRT-Meldungen abgespeichert werden.
Optional. Im Falle eines War-Deployments, werden die Logs nur dann abgespeichert, wenn der Parameter einen Wert besitzt.

BIRT_VIEWER_LOG_LEVEL
Definiert die Stufe der Logging-Funktionalität.
Dabei werden die Standard Java-Bezeichner[5] verwendet: `ALL`, `CONFIG`, `FINE`, `FINER`, `FINEST`, `INFO`, `OFF`, `SEVERE`, `WARNING`.
Der Parameter ist optional, sein Default-Wert auf „`OFF`“ gesetzt.

BIRT_VIEWER_WORKING_FOLDER
Ist ein absoluter Pfad zum Ordner, wo die Report-Designs und temporäre Ordner abgespeichert werden.
Der Parameter ist optional. Eine Report-Design-Datei kann aber nur dann relativ aufgerufen werden, wenn dieser Parameter einen Wert besitzt.

BIRT_VIEWER_SCRIPTLIB_DIR
Ist ein absoluter Pfad zum Verzeichnis, wo Java Event-Handler[6] für BIRT-Reports als Jar-Dateien abgespeichert sind.
Der Parameter ist optional.

BIRT_VIEWER_LOCALE
Der Parameter gibt an, welche Locale für die Reports gelten soll.

[5] Eine ausführliche Information kann man aus der Java-Dokumentation für java.util.logging.Level entnehmen.
[6] sec:BIRTScripting

6.2. Integration von BRV innerhalb einer Web-Anwendung

6.2.1. Zielstellung und Vorgehensweise

Diese praktische Studie hat folgende Aufgaben:

- Integration des BRVs mit Bea WebLogic
- Anpassung der BRV-Konfiguration an eine vorhandene Web-Applikation-Struktur
- Anpassung einiger Layout-Elemente vom BRV nach eigenen Vorgaben

Basis-Infrastruktur
Die zu entwickelnde Reporting-Applikation wird in der folgenden Umgebung laufen:

- Application Server: Bea WebLogic 8
- Server Betriebssystem: Linux SLES 9
- BIRT-Framework und BIRT Report Viewer in der Version 2.2M5
- Datenbanksystem: Oracle 10g
- Java SDK 1.4.2

Vorgehensweise
Die folgenden Arbeitsschritte werden für die BRV-Integration vorgenommen:

1. BRV-Verzeichnisse und Dateien anpassen
2. Die iText Jar-Datei kopieren
3. JDBC Treiber kopieren
4. War-Distribution der Applikation bauen
5. War-Distribution unter Bea WebLogic installieren
6. Web-Applikation testen

6.2.2. Arbeitsschritt 1: BRV-Verzeichnisse und Dateien anpassen

Ursprünglich sieht die Original-BRV-Verzeichnisstruktur wie folgt aus:

Ordner		Beschreibung
Hilfsverzeichnisse von BRV		
logs	⟶	Log-Dateien mit BIRT-Meldungen
report	⟶	Report-Design-Dateien
scriptlib	⟶	Java Event-Handler[7] für BIRT-Reports als Jar-Dateien
BRV-Verzeichnisse		
webcontent	⟶	Statische Inhalte und JSP-Dateien
birt		
ajax	⟶	JavaScript Dateien, werden von Viewer verwendet (u.a. für die asynchrone Kommunikation mit der BIRT Engine und für GUI-Elemente)
images	⟶	Bilder und Icons
pages	⟶	JSP Fragmente, werden bei Aufbau von Viewer verwendet
styles	⟶	CSS Dateien
Web-inf		
lib	⟶	Jars, die BRV benötigt
platform		
configuration	⟶	Konfigurationsdateien für OSGi (s. *Abschnitt 5.2.2*)
plugins	⟶	BIRT Runtime Plug-Ins (s. *Abschnitt 5.4*)
tlds	⟶	Beschreibungen der Tag-Libraries von BRV

Eine Applikation, in welche BIRT integriert werden sollte, besitzt eine ähnliche Struktur:

Ordner		Beschreibung
html	⟶	Statische Inhalte und JSP-Dateien
Web-inf		
classes	⟶	Java-Klassen der Web-Applikation
lib	⟶	Jars, welche die Web-Applikation benötigt
tld	⟶	Tag-Libraries der Web-Applikation

Tabelle 6.2.: BRV Verzeichnisstruktur

Ordner
Hilfsverzeichnisse von BRV
/lfs/wwwmnt/xsapp3-i-app1/t-reports/birt_resources/report/images
/lfs/wwwmnt/xsapp3-i-app1/t-reports/birt_resources/report
/lfs/wwwmnt/xsapp3-i-app1/t-reports/birt_resources/scriptlib
/lfs/wwwmnt/xsapp3-i-app1/logs/birt
Verzeichnisse für die War-Distribution
html
webcontent
birt
ajax
images
pages
styles
Web-inf
classes
lib
platform
configuration
plugins
tld

Notwendige Strukturänderungen:

- Um die Verwaltung der Reports unabhängig von dem Lebenszyklus der Reporting-Applikation durchführen zu können, müssen die Ordner `report` und `scriptlib` von der War-Distribution entkoppelt werden.
- Zusätzlich sollte das Verzeichnis `images` erstellt werden (s. *Abschnitt 6.1.4*).
- Sowohl die Log-Dateien von BRV als auch die Log-Dateien der eigenen Web-Applikation sollten im gleichen Verzeichnis abgespeichert werden.

- Die Ordner mit den statischen Inhalten des BRVs und der eigenen Web-Applikation sollen gesondert behandelt werden. Dies ermöglicht eine unabhängige Release-Planung der beiden Applikationsteile.
- Tag-Libraries von BRV werden aus dem Verzeichnis `tlds` in das Verzeichnis `tld` verschoben.
- Andere Verzeichnisse innerhalb von `Web-inf` sind gleich und können zusammengeschlossen werden.

Die Ziel-Verzeichnisstruktur, die nach den Änderungen entsteht, zeigt die *Tabelle 6.2*.

Um die Struktur-Änderungen dem BRV bekannt zu machen, muss die Konfigurationsdatei `Web.xml` angepasst werden. Das *Listing 6.1* beinhaltet die notwendigen Änderungen an der `Web.xml` Datei:

- Die *Zeilen 1 – 46* beschreiben Hilfsverzeichnisse von BRV.
- Die *Zeilen 61 – 73* definieren Servlets für die BIRT Engine und das BIRT Report Viewer.
- Die *Zeilen 81 – 100* bilden diese Servlets auf URL-Strings ab.
- Die *Zeilen 75 – 78* definieren die Tag-Libraries von BRV.

Listing 6.1: Web.xml: Code für die Integration von BRV

```
  <context-param>
    <param-name>WORKING_FOLDER_ACCESS_ONLY</param-name>
    <param-value>false</param-value>
  </context-param>
  <context-param>
    <param-name>BIRT_VIEWER_IMAGE_DIR</param-name>
    <param-value>
      /lfs/wwwmnt/xsapp3-i-app1/t-reports/birt_resources/report/images/
    </param-value>
  </context-param>
  <context-param>
    <param-name>BIRT_VIEWER_MAX_ROWS</param-name>
    <param-value></param-value>
  </context-param>
  <context-param>
    <param-name>BIRT_VIEWER_DOCUMENT_FOLDER</param-name>
    <param-value></param-value>
  </context-param>
```

```
<context-param>
  <param-name>BIRT_VIEWER_LOCALE</param-name>
  <param-value>en-US</param-value>
</context-param>
<context-param>
  <param-name>BIRT_VIEWER_WORKING_FOLDER</param-name>
  <param-value>
    /lfs/wwwmnt/xsapp3-i-app1/t-reports/birt_resources/report/
  </param-value>
</context-param>
<context-param>
  <param-name>BIRT_VIEWER_LOG_DIR</param-name>
  <param-value>/lfs/wwwmnt/xsapp3-i-app1/logs/birt/</param-value>
</context-param>
<context-param>
  <param-name>BIRT_VIEWER_LOG_LEVEL</param-name>
  <param-value>WARNING</param-value>
</context-param>
<context-param>
  <param-name>BIRT_VIEWER_SCRIPTLIB_DIR</param-name>
  <param-value>
    /lfs/wwwmnt/xsapp3-i-app1/t-reports/birt_resources/scriptlib/
  </param-value>
</context-param>
<context-param>
  <param-name>BIRT_OVERWRITE_DOCUMENT</param-name>
  <param-value>true</param-value>
</context-param>

...
<listener>
  <listener-class>
    org.eclipse.birt.report.listener.ViewerHttpSessionListener
  </listener-class>
</listener>
<listener>
  <listener-class>
    org.eclipse.birt.report.listener.ViewerServletContextListener
  </listener-class>
</listener>
```

```
...
<servlet>
  <servlet-name>EngineServlet</servlet-name>
  <servlet-class>
    org.eclipse.birt.report.servlet.BirtEngineServlet
  </servlet-class>
</servlet>
<servlet>
  <servlet-name>ViewerServlet</servlet-name>
  <servlet-class>
    org.eclipse.birt.report.servlet.ViewerServlet
  </servlet-class>
</servlet>

...
<taglib>
  <taglib-uri>/birt.tld</taglib-uri>
  <taglib-location>/WEB-INF/tld/birt.tld</taglib-location>
</taglib>

...
<servlet-mapping>
  <servlet-name>ViewerServlet</servlet-name>
  <url-pattern>/frameset</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>EngineServlet</servlet-name>
  <url-pattern>/download</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>EngineServlet</servlet-name>
  <url-pattern>/parameter</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>ViewerServlet</servlet-name>
  <url-pattern>/run</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>EngineServlet</servlet-name>
  <url-pattern>/preview</url-pattern>
</servlet-mapping>
```

6.2.3. Arbeitsschritt 2: Die iText Jar-Datei kopieren

Die iText Komponente ist für das Generieren von PDF-Dateien verantwortlich (ausführliche Informationen sind in (Lowagie, 2007) zu finden). Sie wird nicht mit BIRT mitgeliefert und muss extra heruntergeladen[8] und in das Verzeichnis

```
WEB-INF/platform/plugins/com.lowagie.itext/lib
```

kopiert werden.

6.2.4. Arbeitsschritt 3: JDBC Treiber kopieren

Typischerweise werden alle notwendigen JDBC Treiber im Verzeichnis

```
WEB-INF/platform/plugins/org.eclipse.birt.report.data.oda.jdbc_2.2.0.v20070403/↘
    ⟶drivers/
```

abgelegt.

6.2.5. Arbeitsschritt 4: Die War-Distribution der Applikation bauen

Eine War-Datei kann mit folgendem Kommando erstellt werden:

```
jar- cvf <Name>.jar <Verzeichnis-Name>
```

Im Regelfall wird aber eine War-Distribution der Applikation mit Hilfe eines Ant-Skriptes gebaut.

6.2.6. Arbeitsschritt 5: War-Distribution unter Bea WebLogic installieren

Die entsprechende Vorgehensweise beim Deployment ist der Bea Dokumentation zu entnehmen (s. (BEA, 2007)).

6.2.7. Arbeitsschritt 6: Web-Applikation testen

Ein Test-Report `Test.rptdesign` befindet sich in dem vom Parameter `BIRT_VIEWER_WORKING_FOLDER` definierten Verzeichnis `report` (s. *Abschnitt 6.1.3*). Er wird über den folgenden URL-String aufgerufen:

```
http://host:<port>/<report-app-name>/frameset?__report=Test.rptdesign
```

Ist die Ausgabe fehlgeschlagen, können Debug-Ausgaben aus den Log-Dateien bei der Fehleranalyse weiter helfen. Die aktuellsten Meldungen werden beispielsweise mit dem Kommando

[8] Für BIRT 2.2M5 ist die Version 1.4.6 von iText notwendig. Sie ist unter http://prdownloads.sourceforge.net/itext/itext-1.4.6.jar frei verfügbar.

```
tail -f  /lfs/wwwmnt/xsapp3-i-app1/logs/birt/logDatei.log
```

aufgerufen.

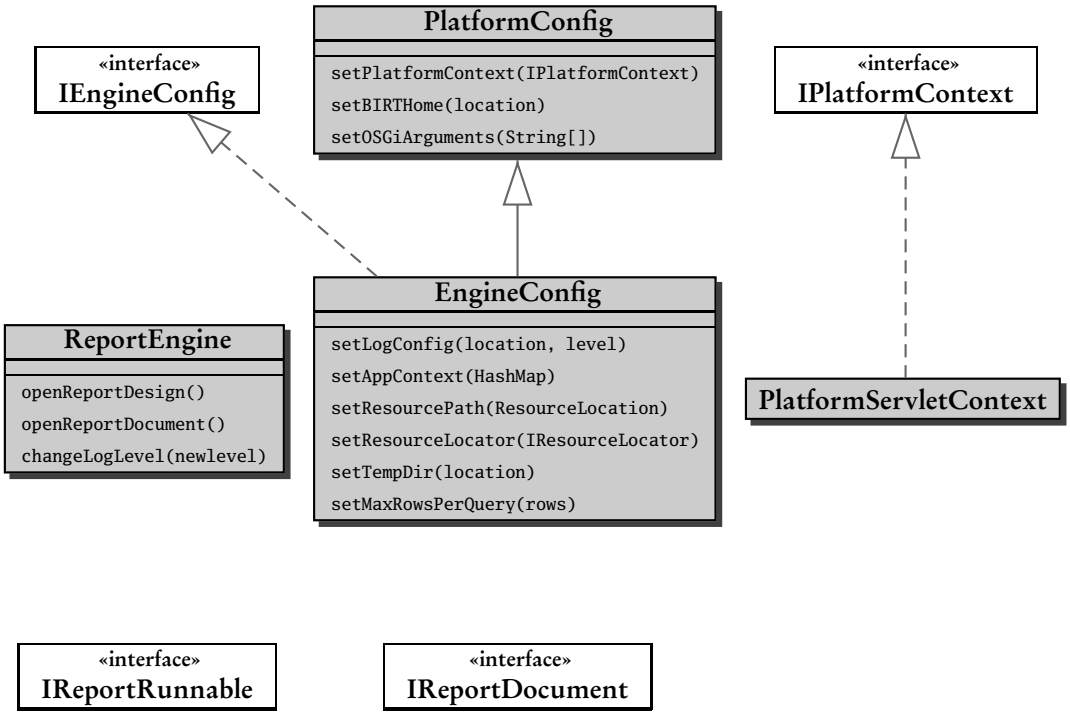
6.3. Entwicklung einer Reporting-Applikation mit BIRT Report Engine APIs

Die BIRT Report Engine (s. *Abschnitt 5.4.2*) kann verwendet werden, um innerhalb einer Java-(Web-)Applikation die Reports aus den Reports Design Dateien generieren zu können.

6.3.1. BIRT Report Engine APIs

BIRT beinhaltet hunderte von Klassen und Interfaces, viele davon sind nur für die BIRT-Committer von Interesse. Für die Report-Entwickler sind vor allem Kenntnisse folgender Klas-

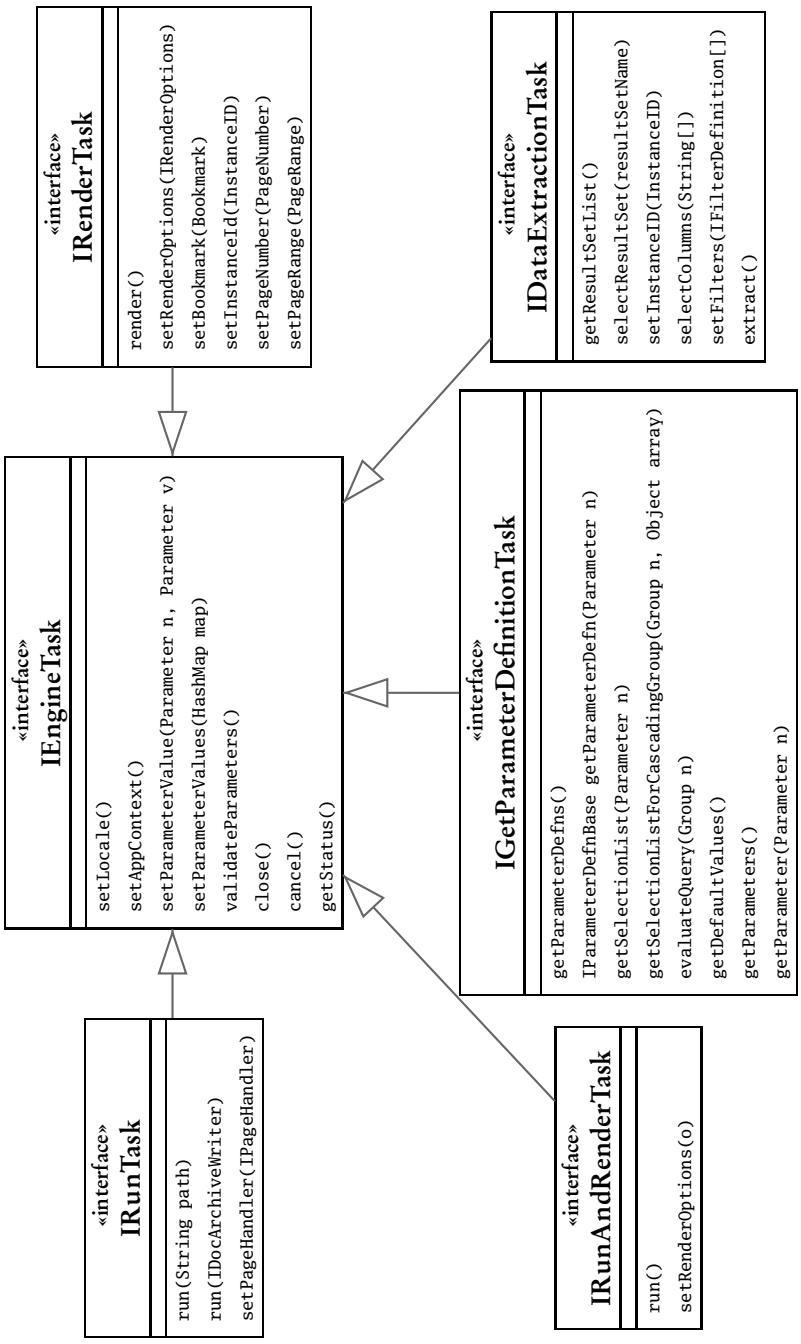
Abbildung 6.2.: Report Engine API und OSGi-Plattform



sen vom großem Nutzen: `ReportEngine`, `EngineConfig`, `IReportRunnable` sowie `IEngineTask` und deren Unterklassen. Die Abbildungen 6.2 und 6.3 zeigen die zugehörigen UML-Klassendiagramme in einer sehr vereinfachten Form. Eine ausführliche Beschreibung der Klassen und

Schnittstellen findet man in der online BIRT-Dokumentation.[9]

Abbildung 6.3.: Report Engine API



[9] Unter BIRT Report Designer ist die Dokumentation unter dem Pfad „Help → Help Contents → BIRT Programmer Reference → Reference → API Reference“ zu finden.

6.3.2. Reports mit BIRT Engine generieren

Eine minimale Applikation, die Reports generiert, kann nach folgendem Muster aufgebaut werden:

1. Report Engine erstellen und konfigurieren
2. Eine Report-Design- oder eine Report-Document-Datei öffnen
3. Report-Parameter abfragen (optional)
4. Das Generieren des Reports vorbereiten
5. Daten aus einer Datenquelle abfragen
6. Report in einem Ausgabe-Format generieren
7. Tasks beenden und die Engine abschließen

Die einzelnen Arbeitsschritte werden im Folgenden ausführlich beschrieben und mit Code-Beispielen verdeutlicht.

6.3.3. Arbeitsschritt 1: Report Engine erstellen und konfigurieren

ReportEngine
Die BIRT Report Engine (RE) ist eine Objekt-Instanz der Klasse `ReportEngine` (s. *Abbildung 6.2*). Sie ist eine Hauptkomponente jeder Reporting-Applikation. Die RE steuert über ihre Tasks das Erstellen eines Reports. Zusätzlich bietet sie die Getter-Funktionen für die Report-Design-Dateien, die Struktur eines Reports und seine Parameter. Die RE ist für die folgenden Aufgaben (Tasks) verantwortlich:

- Konfigurations-Objekt zurückliefern
- Report Design oder Report Document öffnen
- Task erstellen, um Parameter-Definitionen zurückzugeben oder neue Parameter-Werte zu setzen
- Task erstellen, um auf die Daten aus einem Data Set zugreifen zu können
- Report-Format und seinen MIME-Typ bestimmen
- Task erstellen, um einen Report auszuführen bzw. einen Report im bestimmten Format präsentieren

- Task erstellen, um Daten aus einem Report Document extrahieren
- Task erstellen, um die Arbeitsprozesse der RE sauber zu beenden

In einer Applikation wird immer nur eine einzige RE-Instanz erstellt (initialisiert) – die Vorgehensweise entspricht dem Singleton-Pattern (s. Gamma u. a. (1995)). Eine RE kann mehrere Reports aus mehreren Report-Designs erstellen und ist thread-safe. Um die Arbeit mit der RE korrekt abzuschließen, muss die Applikation die `shutdown()` Methode aufrufen: Die Engine und ihre Erweiterungen werden beendet und alle temporären Dateien werden gelöscht.

EngineConfig
Eine Konfiguration für die RE beinhaltet das `EngineConfig`-Objekt (s. *Abbildung 6.2*). Folgende Einstellungen können vorgenommen werden:

- Die Bilder-Ausgabe in einem HTML-Dokument konfigurieren
- Den Logging-Level setzen
- Den Plattform-Kontext anpassen: Applikations- und RE-Verzeichnisse (Engine Home) setzen. Der korrekte absolute Pfad zum Engine Home Verzeichnis ist notwendig, damit die RE überhaupt ausgeführt werden kann
- Ein Verzeichnis für die Libraries und anderer Resources setzen
- Die Scripting-Konfiguration anpassen

Der ReportEngine-Konstruktor erwartet ein EngineConfig Objekt als Argument. Eine Default-Konfiguration wird verwendet, wenn das Objekt null ist. In diesem Fall muss die Ablaufumgebung über eine korrekt gesetzte `BIRT_HOME` Variable verfügen.

Beispiel Das *Listing 6.2* zeigt, wie die Report Engine initialisiert wird. Im Code wird ein `BIRT_HOME` gesetzt und als Ausgabe-Format wird HTML bestimmt.

Listing 6.2: Report Engine erstellen und konfigurieren

```
// EngineConfig Objekt erstellen
EngineConfig config = new EngineConfig();

// Den Pfad zur RE setzen
config.setEngineHome("C:/Program Files/birt-runtime-2_2_0/ReportEngine");

// Den Kontext für eine Stand-alone-Application setzen
IPlatformContext context = new PlatformFileContext();
```

```
config.setEngineContext(context);

// Den HTML-Emitter konfigurieren
HTMLCompleteImageHandler imageHandler = new HTMLCompleteImageHandler();
HTMLEmitterConfig hc = new HTMLEmitterConfig();
hc.setImageHandler(imageHandler);
config.setEmitterConfiguration(RenderOptionBase.OUTPUT_FORMAT_HTML, hc);

// Engine konfigurieren
ReportEngine engine = new ReportEngine(config);
```

6.3.4. Arbeitsschritt 2: Report Design oder Report Document öffnen

Die RE kann die Reports sowohl aus einer Design-Beschreibung als auch aus einer Report-Dokument-Datei generieren (s. *Abschnitt 5.4.4*).

Eine Report-Design-Datei wird vom Objekt des Types `IReportRunnable`, repräsentiert eine Report-Dokument-Datei vom Objekt des Types `IReportDocument` (s. *Abbildung 6.2*).

IReportRunnable

Das Objekt `IReportRunnable` stellt Getter-Methoden für die grundlegenden Eigenschaften eines Reports zur Verfügung. Die Property-Namen sind statische String-Felder und werden im IReportRunnable-Interface definiert.

Beispiel Das *Listing 6.3* zeigt, wie eine Report-Design-Datei geöffnet wird. Wenn die Datei nicht existiert, sollte die Engine geschlossen werden. Des Weiteren wird die Report-Eigenschaft „Author“ abgefragt. Die Variable engine ist ein ReportEngine-Objekt.

Listing 6.3: Report Design öffnen

```
String designName = "./Beispiel.rptdesign";
IReportRunnable runnable = null;
try {
  runnable = engine.openReportDesign(designName);
}
catch (EngineException e)
{
  System.err.println ("Design " + designName + " not found!");
  engine.shutdown();
  System.exit(-1);
}
String author = (String) runnable.getProperty(IReportRunnable.AUTHOR);
```

IReportDocument
Das Objekt `IReportDocument` verwaltet Report-Daten und Struktur. Über seine Methoden kann man u. a. das Report-Inhaltsverzeichnis (TOC), die Bookmarks und Seiten-Informationen anfordern.

Beispiel Das *Listing 6.4* zeigt, wie eine Report-Document-Datei geöffnet wird und wie man die TOC aufruft und darin navigiert.

Listing 6.4: Report Document öffnen

```
String dName = "./SimpleReport.rptdocument";
IReportDocument doc = null;
try {
    doc = engine.openReportDocument(dName);
} catch (EngineException e) {
    System.err.println("Document " + dName + " not found!");
  engine.shutdown();
  System.exit(-1);
}

// TOC holen
TOCNode td = doc.findTOC(null);
java.util.List children = td.getChildren();
long pNumber;
// TOC durchlaufen
if (children != null && children.size() > 0) {
  for (int i = 0; i < children.size(); i++)   {
    // Den nötigen TOC-Eintrag (103) finden
    TOCNode child = (TOCNode) children.get(i);
    if (child.getDisplayString().equals("103")) {
      // Die entsprechende Seiten-Nummer abfragen
      pNumber = doc.getPageNumber(child.getBookmark());
      System.out.println("Page to print is " + pNumber);
    }
  }
}
```

6.3.5. Arbeitsschritt 3: Report-Parameter abfragen

Bevor die Applikation einen Report generiert, kann ein Input an den Report im Form der Parameter erfolgen. Die einfachen Parameter bestehen aus einem Namen und einem zugehörigen

Wert. Eine Reporting-Applikation fordert den Benutzer auf, die Werte für die definierten Parameter einzugeben. Danach werden die Werte an die RE weitergeleitet, um den Report zu generieren.

Beispiel Das *Listing 6.5* zeigt, wie Parameter-Werte gesetzt werden. Diese Parameter werden in einem HashMap-Objekt abgespeichert. Die Variable engine ist ein IReportRunnable-Objekt.

Listing 6.5: Report-Parameter abfragen

```
// Parameter definition Task erstellen
IGetParameterDefinitionTask task = engine.createGetParameterDefinitionTask(runnable);

// Ein Parameter initialisieren
IScalarParameterDefn param = (IScalarParameterDefn)
task.getParameterDefn("customerID");

// Den Default-Parameter-Wert holen
int customerID = ((Double) task.getDefaultValue(param)).intValue();

// Den neuen Parameter-Wert aus GUI bzw. Datenbank holen
...
// Der Wert befindet sich in der Variable inputValue

// Den neuen Parameter-Wert setzen
task.setParameterValue("customerID", inputValue);

// Alle Parameter-Werte abfragen
HashMap parameterValues = task.getParameterValues();
```

Die Verwaltung der Report-Parameter ist nicht trivial. Ausführliche Beschreibung zu allen Parameter-Typen sowie Beispiele sind unter Weathersby u. a. (2006) zu finden.

6.3.6. Arbeitsschritt 4: Das Generieren des Reports vorbereiten

Die folgenden Engine-Tasks können zum Generieren eines Reports verwendet werden:

createRunAndRenderTask
Die `createRunAndRenderTask()` erstellt aus einer Report-Design-Datei ein `IRunAndRenderTask`-Objekt (s. *Abbildung 6.3*). Dieses Objekt kann dann zum Generieren einer einfachen HTML-Ausgabe verwendet werden.

createRunTask
Die `createRunTask()` erstellt aus einer Report-Design-Datei ein `IRunTask`-Objekt (s. *Abbildung 6.3*). Dieses Objekt kann dann zum Generieren einer Report-Dokument-Datei (`.rptdocument`) verwendet werden.

createRenderTask
Die `createRenderTask()` erstellt aus einer Report-Dokument-Datei ein `IRenderTask`-Objekt (s. *Abbildung 6.3*). Dieses Objekt kann dann zum Generieren eines Reports (komplexe HTML- oder PDF-Ausgabe) verwendet werden.

Bevor die Reports generiert werden können, muss die Applikation einige Ausgabe-Eigenschaften setzen. Zum Ersten definiert man, was für eine Ausgabe gefordert wird: Ausgabe in eine Datei oder als Data-Stream. Zum Zweiten, wie die HTML-Ausgabe aussehen soll: Eine komplette HTML-Ausgabe oder eine Ausgabe ohne HTML-Headers zum Einfügen innerhalb einer vorhandenen HTML-Seite – *Embeddable HTML*.

Beispiel Das *Listing 6.6* zeigt, wie die IRunAndRenderTask verwendet wird und wie Parameter und Ausgabe-Optionen gesetzt werden. Die Variable `name` ist der Name des reports.

Listing 6.6: Das Generieren des Reports vorbereiten

```
// createRunAndRenderTask starten
IRunAndRenderTask task = engine.createRunAndRenderTask(runnable);

// Parameter-Werte setzen
task.setParameterValues(parameterValues);

// Parameter-Werte validieren
boolean parametersAreGood = task.validateParameters();

// Den Namen für die Ausgabe-Datei setzen
HTMLRenderOption options = new HTMLRenderOption();
String output = name.replaceFirst(".rptdesign", ".html");
options.setOutputFileName(output);

// Die Render-Eigenschaften anwenden
task.setRenderOption(options);

// HTML-Rendering-Context initialisieren
HTMLRenderContext renderContext = new HTMLRenderContext();
renderContext.setImageDirectory("image");
```

```
// Den Context der Task zuweisen
HashMap appContext = new HashMap();
appContext.put(EngineConstants.APPCONTEXT_HTML_RENDER_CONTEXT, renderContext);
task.setAppContext(appContext);
```

6.3.7. Arbeitsschritt 5: Daten aus einer Datenquelle abfragen

Die RE kann ein Connection-Objekt zur Datenquelle selbst erstellen oder ein vorhandenes Connection-Objekt der Applikation nutzen. Das ist vor allem in den Web-Applikationen nützlich, wo man oft einen Connection-Pool für solche Zwecke verwendet. Die notwendigen Informationen zur Verbindung und zu Treiber müssen in diesem Fall an die Report Engine weitergegeben werden.

Beispiel Das *Listing 6.6* zeigt, wie eine Data-Source-Verbindung mit einem eigenem Treiber (`mydatapluginname`) aufgebaut werden kann.

Listing 6.7: Eine Data-Source-Verbindung aufbauen

```
HashMap contextMap = new HashMap();
HTMLRenderContext renderContext = new HTMLRenderContext();
contextMap.put(EngineConstants.APPCONTEXT_HTML_RENDER_CONTEXT, renderContext);

// Eine Pool-Connection holen
Connection myConnection = getConnectionFromPool();
contextMap.put("org.eclipse.birt.mydatapluginname", myConnection);
task.setContext(contextMap);
```

6.3.8. Arbeitsschritt 6: Report in einem Ausgabe-Format generieren und die Engine abschließen

Um einen Report zu generieren, wird die Methode run() des Objektes IRunAndRenderTask oder des Objektes IRunTask aufgerufen. Wenn noch weitere Reports generiert werden sollen, kann die Applikation mit dem Arbeitsschritt 2 (s. *Abschnitt 6.3.4*) fortfahren, sonst wird die RE beendet.

Beispiel Im *Listing 6.8* wird ein Report zunächst generiert und dann die RE beendet. Die Variable `task` steht für ein IRunAndRenderTask oder ein IRunTask Objekt, die Variable `output` ist der Name der Ausgabe-Datei.

Listing 6.8: Einen Report generieren und die Engine abschließen

```
try {
  task.run();
  System.out.println("Created Report " + output + ".");
}
catch (EngineException e1) {
  System.err.println("Report " + name + " run failed.");
  System.err.println(e1.toString());
}

engine.shutdown();
```

7. Fazit und Zukunftsaussichten

„Nach dem Spiel ist vor dem Spiel“

Die vorliegende Arbeit bietet eine Beschreibung eines Architektur-Konzeptes für eine Reporting-Applikation. Im Hintergrund stehen die theoretischen Konzepte zu Software-Entwicklung und Software-Architekturen sowie zu Reporting-Prozessen im Unternehmen. Die entwickelte Applikation ist in der Lage, operative bzw. Ad-hoc-Reporting-Anforderungen eines Produktionsunternehmen abzudecken. Sie basiert auf J2EE-Technologien und verwendet das open-source Framework Eclipse BIRT. Die Arbeit und die Entwicklung sind innerhalb eines Projektes in der BMW Group entstanden und weisen praktische Erfahrungen mit einer real gegebenen Problematik nach.

In diesem Kapitel werden die Ergebnisse aufgelistet sowie einige Verbesserungsvorschläge und mögliche Zukunftsaussichten genannt.

7.1. Ergebnisse

Die folgenden Ergebnisse wurden im Rahmen des Projektes und der schriftlichen Arbeit erreicht:

- Die Anforderungen an eine Reporting-Applikation wurden gesammelt.
- Die Software-Architektur für eine Reporting-Architektur wurde konstruiert, beschrieben sowie mit den verantwortlichen Akteuren im Betrieb diskutiert und konsolidiert.
- Eine Einigung und Genehmigung der Lösung auf der Basis Eclipse BIRT wurde erzielt.
- Eine initiale Version der Web-Applikation wurde entwickelt und auf einem Test-System ausgerollt.
- Eine Reporting-Testumgebung wurde eingerichtet. Sie wurde intensiv verwendet und beinhaltete bei Ende der Niederschrift dieser Arbeit bereits mehrere Duzend lauffähige Reports.
- Eine Schulung für die Key-User des Systems aus den betroffenen Werken und Technologien wurde durchgeführt. Die Präsentationen und Lehrmittel wurden den Usern übermittelt.

- Die konstruierte Architektur, Dokumentation, die Code-Basis sowie die gewonnenen Erfahrungen wurden an die IT-Architekten und Verantwortlichen des Unternehmens übergeben.

7.2. Verbesserungsvorschläge und Zukunftsaussichten

Da eine Software und ihre Architektur in iterativen Prozessen entwickelt werden, tauchten einige interessante Erkenntnisse und Ideen erst gegen Ende des Projektes auf. An dieser Stelle sollen daraus abgeleitete Vorschläge zur Verbesserung und Weiterentwicklung der Anwendung kurz aufgelistet werden.

Verbesserungen an der Web-Applikation

- Autorisierungsmechanismen des Frameworks können verbessert werden: Das Konzept soll sich u. a. auch auf die Reports selbst beziehen können.
- Fachkonzepte für die Report-Gruppen können erweitert werden: Eine Report-Gruppe soll u. a. in der Lage sein, die gemeinsamen Parameter ihrer Reports entgegen zu nehmen und zu einer Session den beteiligten Reports zur Verfügung zu stellen. Dadurch kann eine Navigation zwischen unterschiedlichen Reports der gleichen Gruppe ohne eine zusätzliche Parameter-Eingabe erfolgen.
- Die Excel- und Word-Ausgaben, welche mit der neuen BIRT Version entstehen werden werden, können innerhalb der Applikation integriert werden.
- Das Navigation-Menü der Applikation kann mit CSS barrierefrei gestaltet werden.
- Verwaltung der Uploads kann verbessert werden. Evtl. kann eine Versionierung einführt werden.

Verbesserungen an dem Prozess

- Die Verantwortlichkeiten und Rollen-Verteilung bei den Key-User kann optimiert werden. Der Report-Erstellungsprozess könnte nach einen klaren Workflow erfolgen.
- Werksübergreifende Templates und Libraries könnten erstellt werden.
- Alle Report-Design-Dateien könnten innerhalb einer eindeutigen Verzeichnisstruktur an einer zentralen Stelle verwaltet werden. Zusätzlich empfiehlt sich hier eine Versionskontrolle.

Zukunftsaussichten

Die neue Version des BIRT-Framework (Eclipse BIRT 2.2) wird u. a. die fehlenden Funktionen für den Excel-Export und Cross-Tables beinhalten. Ein Update der Applikation auf diese neue BIRT-Version wird empfohlen.

Literaturverzeichnis

Die Literaturangaben sind alphabetisch nach den Namen der Autoren sortiert. Bei mehreren Autoren wird nach dem ersten Autor sortiert.

[Alur u. a. 2003] Alur, Deepak ; Crupi, John ; Malks, Dan: *Core J2EE Patterns. Best Practices and Design Strategies*. 2. Aufl. Sun Microsystems Press, 2003. – 650 S

[Bass u. a. 2003] Bass, Len ; Clements, Paul ; Kazman, Rick: *Software Architecture in Practice*. 2. Aufl. Addison-Wesley, 2003. – 512 S

[Bauer und King 2007] Bauer, Christian ; King, Gavin: *Java Persistence with Hibernate*. Manning, 2007. – 841 S

[BEA 2007] BEA: *Product Documentation*. 2007. – URL `http://edocs.bea.com`

[Böhl u. a. 2003] Böhl, Lars ; Lammert, Uwe ; Molterer, Sascha ; Eisele, Markus: *ITA Quick Guide. Schnellreferenz für die grafische Notation der Sichten*. 2003. – BMW AG Intranet

[Bindbeutel 2006] Bindbeutel, Kurt: *Kriterienliste Reporting-Anwendungen*. 2006. – BMW AG Intranet

[Buschmann u. a. 1996] Buschmann, Frank ; Meunier, Regine ; Rohnert, Hans ; Sommerlad, Peter: *A System of Patterns. Pattern-Oriented Software Architecture*. Wiley, 1996. – 476 S

[Chamoni und Gluchowski 2006a] Chamoni, Peter ; Gluchowski, Peter: Analytische Informationssysteme - Einordnung und Überblick. In: Chamoni, Peter (Hrsg.) ; Gluchowski, Peter (Hrsg.): *Analytische Informationssysteme. Business Intelligence-Technologien*. 2. Aufl. Springer, 2006, S. 2 – 22

[Chamoni und Gluchowski 2006b] Chamoni, Peter (Hrsg.) ; Gluchowski, Peter (Hrsg.): *Analytische Informationssysteme. Business Intelligence-Technologien*. 3. Aufl. Springer, 2006. – 458 S

[Clayberg und Rubel 2006] CLAYBERG, Eric ; RUBEL, Dan: *Eclipse: Building Commercial-Quality Plug-ins*. 2. Aufl. Addison-Wesley, 2006 (The Eclipse Series). – 864 S

[Cognos 2007] COGNOS: *Cognos 8 Business Intelligence. Reporting*. 2007. – URL `http://www.cognos.com/products/cognos8businessintelligence/reporting.html`

[Daum 2007] DAUM, Berthold: *Rich-Client-Entwicklung mit Eclipse 3.2. Anwendungen Entwickeln mit der Rich Client Platform*. 2. Aufl. dpunkt.verlag, 2007. – 500 S

[Düsing 2006] DÜSING, Roland: Knowledge Discovery in Databases. Begriff, Forschungsgebiet, Prozess und System. In: *Analytische Informationssysteme. Business Intelligence-Technologien*. 2. Aufl. Springer, 2006

[Eckerson 2002] ECKERSON, Wayne W.: *The rise of analytic applications: Build or buy? TDWI Report Series 101*. 2002

[Eclipse 2007a] ECLIPSE: *BIRT Newsgroup*. 2007. – URL `news://news.eclipse.org/eclipse.birt`

[Eclipse 2007b] ECLIPSE: *BIRT Project Page*. 2007. – URL `http://www.eclipse.org/birt/phoenix`

[Flanagan 2006] FLANAGAN, David: *JavaScript. The Definitive Guide*. 5. Aufl. O'Reilly, 2006. – 995 S

[Fowler 2003] FOWLER, Martin: Who Needs an Architect? In: *Software, IEEE* Volume 20 (2003), 9, Nr. 5, S. 11 – 13

[Fowler u. a. 2002] FOWLER, Martin ; RICE, David ; FOEMMEL, Matthew: *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002. – 560 S

[Freeman u. a. 2004] FREEMAN, Eric ; FREEMAN, Elisabeth ; SIERRA, Kathy: *Head First Design Patterns*. O'Reilly, 2004. – 676 S

[Friedman und Hostmann 2004] FRIEDMAN, Ted ; HOSTMANN, Bill: *The Cornerstones of Business Intelligence Excellence*. April 2004

[Gamma u. a. 1995] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph E.: *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. – 395 S

[Gluchowski 2006] GLUCHOWSKI, Peter: Techniken und Werkzeuge zum Aufbau betrieblicher Berichtssysteme. In: CHAMONI, Peter (Hrsg.) ; GLUCHOWSKI, Peter (Hrsg.): *Analytische Informationssysteme. Business Intelligence-Technologien*. 2. Aufl. Springer, 2006, S. 207 – 226

[Gómez u. a. 2006] GÓMEZ, Jorge M. ; RAUTENSTRAUCH, Claus ; CISSEK, Peter ; GRAHLHER, Björn: *Einführung in SAP Business Information Warehouse*. Springer, 2006. – 273 S

[Heffelfinger 2006] HEFFELFINGER, David R.: *JasperReports for Java Developers. Create, Design, Format, and Export Reports with the World's Most Popular Java Reporting Library*. Packt Publishing, 2006. – 328 S

[Hrushchak 2006] HRUSHCHAK, Ruslan: *Projektauftrag für die Konzeption und Entwicklung einer werksübergreifenden Analyse- und Reporting-Komponente für IPS-T*. Oktober 2006. – Projektforum BWM Group

[Hrushchak und Priemuth 2006] HRUSHCHAK, Ruslan ; PRIEMUTH, Thomas: *IT Prozess Quality Management. Überblick und Grundlagen vom Qualitätsmanagement-System für IT in der BMW Group*. 2006

[Husted u. a. 2003] HUSTED, Ted ; DUMOULIN, Cedric ; FRANCISCUS, George ; WINTERFELDT, David: *Struts in Action.Building web applications with the leading Java framework*. Manning, 2003. – 630 S

[IEEE 2000] IEEE: *IEEE Recommended Practice for Architectural Description for Software-Intensive Systems. IEEE Std 1471-2000*. IEEE, 2000

[Inmon 2002] INMON, William H.: *Building the Data Warehouse*. 3. Aufl. Wiley, 2002. – 356 S

[ITIL 2006] ITIL: *ITIL. Information Technology Infrastructure Library*. 2006. – URL `http://www.itil.co.uk`

[ITPM 2007] ITPM: *ITPM (IT Prozess Quality Management). BMW AG Intranet*. 2007. – BMW AG Intranet

[Johnson 1997] JOHNSON, Ralph E.: Frameworks = (components + patterns). In: *Communications of the ACM* Volume 40 (1997), October, Nr. 10, S. 39 – 42

[Kemper und Finger 2006] KEMPER, Hans-Georg ; FINGER, Ralf: Transformation operativer Daten. Konzeptionelle Überlegungen zur Filterung, Harmonisierung, Aggregation und Anreicherung im Data Warehouse. In: CHAMONI, Peter (Hrsg.) ; GLUCHOWSKI, Peter (Hrsg.): *Analytische Informationssysteme. Business Intelligence-Technologien*. 2. Aufl. Springer, 2006, S. 113 – 128

[Kemper u. a. 2004] KEMPER, Hans-Georg ; MEHANNA, Walid ; UNGER, Carsten: *Business Intelligence - Grundlagen und praktische Anwendungen*. 2. Aufl. Vieweg, 2004. – 223 S

[Lowagie 2007] Lowagie, Bruno: *iText in Action. Creating and manipulating PDF*. Manning, 2007. – 657 S

[Maier und Rechtin 2000] Maier, Mark W. ; Rechtin, Eberhardt: *The Art of Systems Architecture*. 2. Aufl. CRC Press, 2000. – 344 S

[McAffer und Lemieux 2005] McAffer, Jeff ; Lemieux, Jean-Michel: *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java(TM) Applications*. Addison-Wesley, 2005 (The Eclipse Series). – 552 S

[McConnell 2004] McConnell, Steve: *Code Complete. A Practical Handbook of Software Construction: A Practical Handbook of Software Costruction*. 2. Aufl. Microsoft Press, 2004. – 960 S

[Meyer 1997] Meyer, Bertrand: *Object-oriented Software Construction*. 2. Aufl. Prentice Hall International, 1997. – 1296 S

[Molterer und Eisele 2003] Molterer, Sascha ; Eisele, Markus: *ITA in a Nutshell. Architektur-Dokumentation im Anwendungsprojekt*. 2003. – BMW AG Intranet

[Peh u. a. 2006] Peh, Diana ; Hannemann, Alethea ; Hague, Nola: *BIRT: A Field Guide to Reporting*. Addison-Wesley, 2006. – 666 S

[Polke 2001] Polke, Martin: *Prozeßleittechnik*. 2. Aufl. Oldenburg, 2001. – 948 S

[ProClarity 2005] ProClarity: *Blending Reporting and Analytics: Putting the Decision-Maker First*. July 2005

[Rhino 2007] Rhino: *Mozilla Rhino Project Page*. 2007. – URL `http://www.mozilla.org/rhino`

[Rosenbaum 2007] Rosenbaum, Scott: A Primer On ODA Extensions and BIRT. Creating Custom Data Sources for BIRT Reports Using ODA Extension Points. In: *Eclipse Magazine* 8 (2007), S. 30–38

[Scherp und Boll 2006] Scherp, Ansgar ; Boll, Susanne: Framework-Entwurf. In: Reussner, Ralf (Hrsg.) ; Hasselbring, Wilhelm (Hrsg.): *Handbuch der Software-Architektur*. dpunkt.verlag, 2006, S. 395–417

[Schmidt u. a. 2004] Schmidt, Douglas C. ; Gokhale, Anirudda ; Natarajan, Balachandran: Frameworks: Why They are Important and How to Apply Them Effectively. In: *ACM Queue magazine* Volume 2 (2004), July/August, Nr. 5

[Schulz 2006] SCHULZ, Jeannette: *Evaluation Reporting Frameworks*. April 2006. – BMW AG Intranet

[Simon 1996] SIMON, Herbert: *The Sciences of the Artificial*. 3. Aufl. MIT Press, 1996. – 215 S

[Software Engineering Institute 2007] SOFTWARE ENGINEERING INSTITUTE, Carnegie Mellon U.: *How Do You Define Software Architecture?* 2007. – URL `http://www.sei.cmu.edu/architecture/definitions.html`

[Starke 2005] STARKE, Gernot: *Effektive Software-Architekturen. Ein praktischer Leitfaden*. 2. Aufl. Hanser, 2005. – 290 S

[Szyperski 2002] SZYPERSKI, Clemens: *Component Software. Beyond Object-Oriented Programming*. 2. Aufl. Addison-Wesley, 2002. – 589 S

[Venkatraman und Weathersby 2007] VENKATRAMAN, Krishna ; WEATHERSBY, Jason: Getting Expressive with BIRT. Using the BIRT Expression Builder for Report Customization. In: *Eclipse Magazine* 2 (2007), S. 29–35

[Weathersby 2006] WEATHERSBY, Jason: *Deploying BIRT*. 07 2006. – URL `http://www.onjava.com/pub/a/onjava/2006/07/26/deploying-birt.html`

[Weathersby 2007] WEATHERSBY, Jason: Deploying the BIRT Viewer to JBoss. In: *Eclipse Magazine* 6 (2007), S. 38–43

[Weathersby u. a. 2006] WEATHERSBY, Jason ; FRENCH, Don ; BONDUR, Tom ; TATCHELL, Jane ; CHATALBASHEVA, Iana: *Integrating and Extending BIRT*. Addison-Wesley, 2006. – 568 S

[Willenborg 2000] WILLENBORG, Kai: *Berichtswesen - Vereinigung von Gegensätzen*. 12 2000. – URL `http://www.sapdesignguild.org/editions/edition2/willenborg/index.html`

Verzeichnis der Anhänge

A. IPS-T-Reporting: Anforderungen

Organisatorische Rahmenbedingungen

O 01 Prozess-Spezialisten sind in der Lage, die Reports selbst zu erstellen und freizugeben.

O 011 Für das Erstellen von Reports sind keine Programmierkenntnisse erforderlich.

O 02 Die Entwicklung und das Rollout der Reports dürfen nicht an das Releasemanagement der Applikation gekoppelt sein.

O 03 Die Lösung soll gewährleisten, dass die Reports mindestens zu Büroarbeitszeiten (8/5) verfügbar sind.

O 04 Für das Aufrufen und Generieren von Reports darf keine Lizenzierung erforderlich sein.

O 05 Für das Erstellen von Reports darf keine User-License erforderlich sein. Eine Node-Locked License-Modell wäre vorstellbar.

O 06 Ein neuer Report muss innerhalb von 2 Tagen bereitgestellt werden können. Dies umfasst sowohl die Implementierung als auch den Test.

O 07 Die Auswahl von Report muss personalisiert werden können.

Architekturrelevante Anforderungen

A 01 Die Antwortzeit der Online-Reports soll maximal 5 sec betragen.

A 02 Die Authentisierung und Autorisierung muss für jeden Report frei konfigurierbar sein.

A 03 Die Lösung muss eine Schnittstelle für Anwendungsprogrammierung zur Verfügung stellen.

A 04 Die Lösung muss die Integration mehrerer Datenquellen unterstützen.

A 05 Die Lösung muss mindestens die Sprachen Deutsch und Englisch unterstützen.

A 06 Die Reporting-Funktionalität muss via Web zur Verfügung gestellt werden.

A 07 Der Client zur Erstellung von Reports kann auch als FAT-Client zur Verfügung gestellt werden.

Funktionale Reporting-Anforderungen

F 01 Folgende Reporting-Arten sollen unterstützt werden: Standard- und Online-Reports, Ad-hoc-Reports, Batch-Reporting.

F 02 Die Reporting-Lösung soll über eine grafische Benutzerschnittstelle für das Erstellen von Reports verfügen.

F 021 Die Reporting-Lösung soll eine Unterstützung für Vorlagen und Bibliotheken anbieten.

F 03 Die Anwendung von mehreren unterschiedlichen Datenquellen innerhalb eines einzelnen Reports soll möglich sein. S. Anforderung A 4.

F 031 Die Möglichkeit, eigene Datenquellen zu definieren, soll vorhanden sein.

F 032 Die Möglichkeit, Datenquellen dynamisch – zur Generierungszeit eines Reports – anbinden zu können, sollte vorhanden sein.

F 04 Der Nutzer soll in der Lage sein, mehrere unterschiedliche Abfragen / Darstellungen innerhalb eines einzelnen Reports zu definieren bzw. zu verwenden.

F 05 Es soll möglich sein, eine Beschränkung der maximalen Anzahl der Zeilen im Result Set einer Abfrage zu definieren.

F 06 Es soll möglich sein, die Ergebnisse einer Abfrage als eine Liste, eine Tabelle, oder ein Diagramm darzustellen.

F 07 Die Lösung soll eine Unterstützung für Diagramme (Charts) bieten. S. Anforderung F 06.

F 071 Es sollen u. a. Säulen-, Kuchen-, Flächen- und Linien-Diagramme unterstützt werden.

F 072 Die einzelne Diagramm-Arten sollen miteinander kombinierbar sein. Eine Kombination von unterschiedlichen Diagramm-Typen in einem Chart soll möglich sein.

F 073 Eine Kombination von Chart- und Text-Daten wie Tabellen oder Listen soll innerhalb eines Reports möglich sein.

F 08 Die Charts sind über einen Wizard zu erstellen.

F 09 Die Möglichkeit, eine Anpassung der Abfragen und der Darstellung eines Reports über Parameter zu durchführen, sollte vorhanden sein.

F 091 Die Lösung soll kaskadierende Parameter unterstützen.

F 10 Folgende Funktionen sollen verfügbar sein:

F 101 Sortierung und Gruppierung,

F 102 Aggregation von Daten (Summe, Minimum, Maximum, Mittelwert),

F 103 Filter- und Mapping-Funktionen.

F 11 Die Möglichkeit, eigene selbstdefinierte Funktionen zu verwenden, soll gewährleistet werden.

F 12 Das Layout soll über Styles frei definierbar sein.

F 13 Funktionen zum Erstellen von Master-Seite, Header- / Footer-Definitionen sollen existieren.

F 14 Integration von Bildern soll möglich sein.

F 15 Die Lösung soll über eine Scripting-Schnittstelle verfügen.

F 16 Es sollen Export-Funktionen in die folgenden Daten-Formate existieren:

F 161 HTML,

F 162 PDF,

F 163 CSV,

F 164 MS Excel.

F 17 Die Lösung soll eine umfangreiche und verständliche Benutzer-Dokumentation enthalten.

B. Reporting Framework Evaluierung

Bei der Reporting-Framework-Evaluierung (s. Schulz (2006)) wurden zwei Open-Source Projekte – Eclipse BIRT und Jasper Reports – untersucht und miteinander verglichen.

Zu einer Entscheidung für oder gegen ein Framework kann die tabellarische Übersicht (s. *Tabelle B.1*) der Vor- und Nachteile des jeweiligen Frameworks beitragen. Die Qualitätsmerkmale – Fragestellungen und Kriterien – entstanden aus der Zusammenarbeit von Software-Architekten und Stakeholdern und wurden vor der Evaluierung festgelegt.

Die Bewertungsskala:

- ⋆⋆ voll erfüllt, sehr gut
- ⋆ erfüllt
- – mit Einschränkungen erfüllt, Mängel
- – – nicht erfüllt

Tabelle B.1.: Evaluierung Reporting Frameworks

Qualitätskriterium	BIRT	Jasper
Funktionalität – Report Design / Konfiguration		
Gibt es eine GUI für die Erstellung des Report Designs bzw. der Report Konfiguration?	⋆⋆	⋆
In welcher Form und welchem Format werden die Report Designs bzw. Report Konfigurationen erstellt und abgespeichert?	xml	xml
Wie wird das Layout für die einzelnen Ausgabeformate definiert?	⋆⋆	⋆
Werden mehrere Datenquellen in einem Report unterstützt?	⋆⋆	⋆

Weiter auf der nächsten Seite

Qualitätskriterium	BIRT	Jasper
Sind die Datenquellen zur Laufzeit dynamisch anbindbar?	⋆	⋆
Werden unterschiedliche Typen von Datenquellen unterstützt?	⋆	⋆⋆
Können mehrere Datenanfragen für einen Report definiert werden?	⋆⋆	⋆⋆
Können Abfragen parametriert werden?	⋆⋆	⋆
Können die Report Anfragen unterschiedlich sortiert und gruppiert werden? Wie wird die Sortierung und Gruppierung realisiert?	⋆⋆	⋆⋆
Gibt es eine Möglichkeit, eine benutzerabhängige Filterung der Daten eines Reports vorzunehmen (bedingte WHERE-KLAUSELN)?	⋆⋆	k.A.
Unterstützt das Framework die Erstellung von Charts?	⋆⋆	⋆⋆
Wird die Integration von Bildern in einen Report unterstützt?	⋆⋆	⋆⋆
Können die Daten des Reports nach Absetzten der Datenabfrage bearbeitet werden?	⋆⋆	⋆⋆
Bietet das Framework Unterstützung für Aggregation von Daten unabhängig vom Datenquelltyp?	⋆⋆	⋆⋆
Unterstützt das Framework eine Internationalisierung der Reports?	⋆⋆	⋆⋆
Funktionalität - Ausgabe und Integration		
Wie werden die vordefinierten Reports ausgeführt und gibt es hierfür eine GUI?	⋆⋆	–
Welche Ausgabeformate unterstützt das Reporting Framework?	⋆	⋆⋆
Auf welche Art und Weise ist das Framework in Anwendungen integrierbar?	⋆⋆	–
Erweiterbarkeit		

Weiter auf der nächsten Seite

Qualitätskriterium	BIRT	Jasper
Welche Erweiterungen sind für den Benutzer möglich?	⋆⋆	⋆
Nutzbarkeit/Verständlichkeit		
Welchen Umfang und welche Qualität weist die Dokumentation auf?	⋆⋆	–
Welcher Einarbeitungsaufwand ist nötig, um das Framework zu nutzen?	⋆⋆	–
Wie sieht das Exception Handling des Frameworks aus? Sind die Fehlermeldungen für den Benutzer verständlich und sinnvoll?	⋆	–
Zukunftssicherheit/Standardisierung		
Wie sieht die Verbreitung und Bekanntheit sowie der Einsatz des Frameworks aus?	⋆⋆	⋆⋆
Wie sieht die vergangene und zukünftige Versionsentwicklung für das Framework aus?	⋆⋆	⋆
Wird die Weiterentwicklung von einzelnen Firmen/Organisationen dominiert?	⋆⋆	⋆
Sicherheit/Zuverlässigkeit		
Wer ist der Produktanbieter?	⋆⋆	⋆
Gibt es kommerziellen Support für das Framework?	⋆⋆	⋆⋆
Wie sieht die Qualität der Code-Basis aus?	⋆⋆	⋆

C. Hibernate Mapping-Dateien der Reporting Web-Applikation

Plant.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="com.bmw.xsapps.reporting.objects">
  <class name="Plant" table="plants" lazy="true">
    <id name="id" type="java.lang.Long" unsaved-value="null">
      <generator class="native" />
    </id>
    <property name="name" type="string" length="255" not-null="true"
      unique="true" />
    <property name="description" type="string" length="255" />
    <property name="isLink" type="boolean" not-null="true"
      column="plant_is_link">
    </property>
    <property name="link" type="string" length="255"
      not-null="false" column="link_to_plant" />
    <set name="technologies" inverse="true" cascade="all" lazy="true">
      <key column="plant_id"></key>
      <one-to-many class="Technology" />
    </set>
  </class>
</hibernate-mapping>
```

Technology.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```

```
<hibernate-mapping package="com.bmw.xsapps.reporting.objects">
  <class name="Technology" table="technologies" lazy="true">
    <id name="id" type="java.lang.Long" unsaved-value="null">
      <generator class="native" />
    </id>
    <property name="name" type="string" length="255" not-null="true"
      unique-key="unique_technologies" />
    <property name="description" type="string" length="255" />
    <many-to-one name="plant" column="plant_id" class="Plant"
      not-null="true" unique-key="unique_technologies" lazy="false">
    </many-to-one>
    <set name="report_groups" inverse="true" cascade="all" lazy="true">
      <key column="technology_id"></key>
      <one-to-many class="ReportGroup" />
    </set>
  </class>
</hibernate-mapping>
```

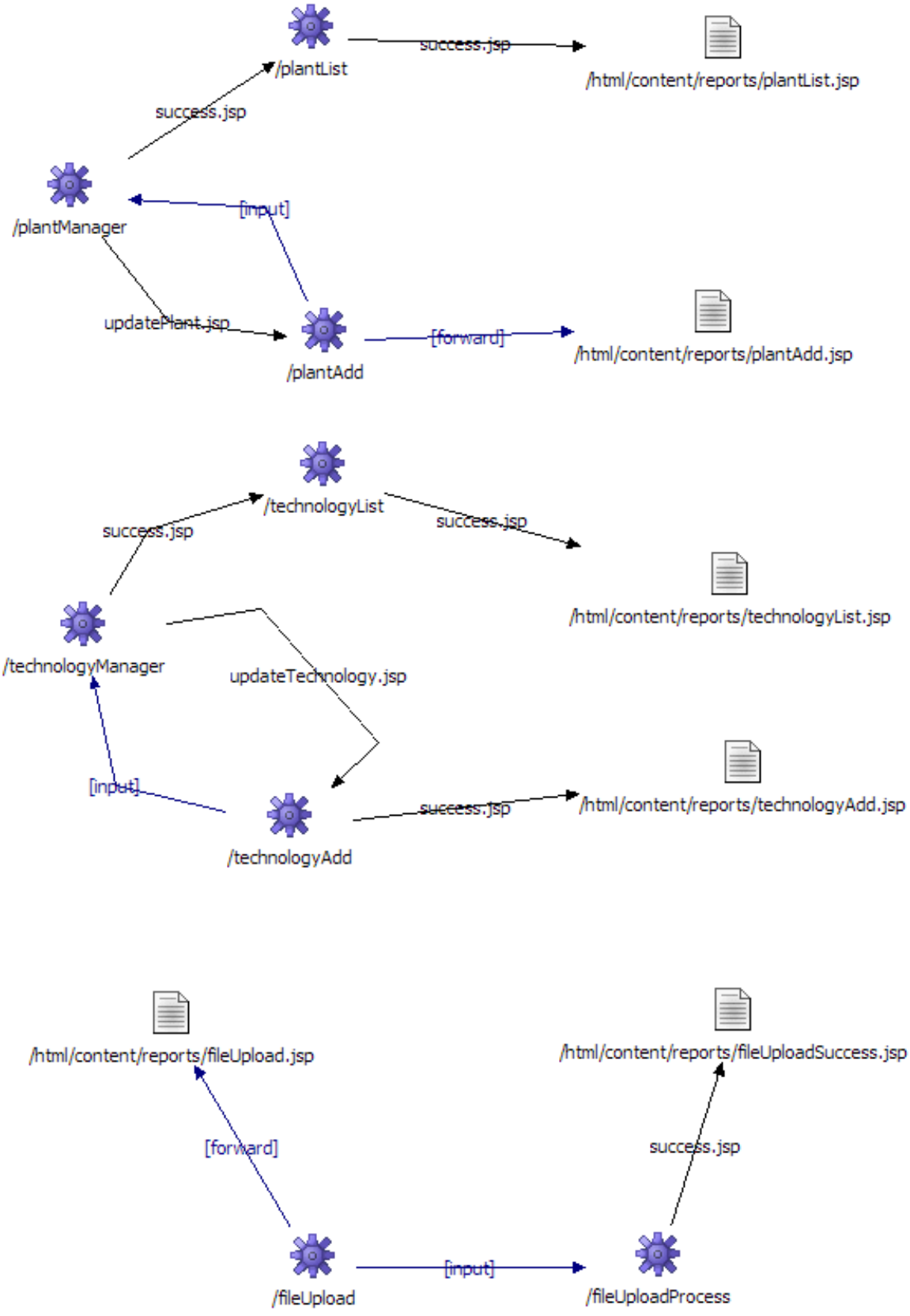
ReportGroup.hbm.xml

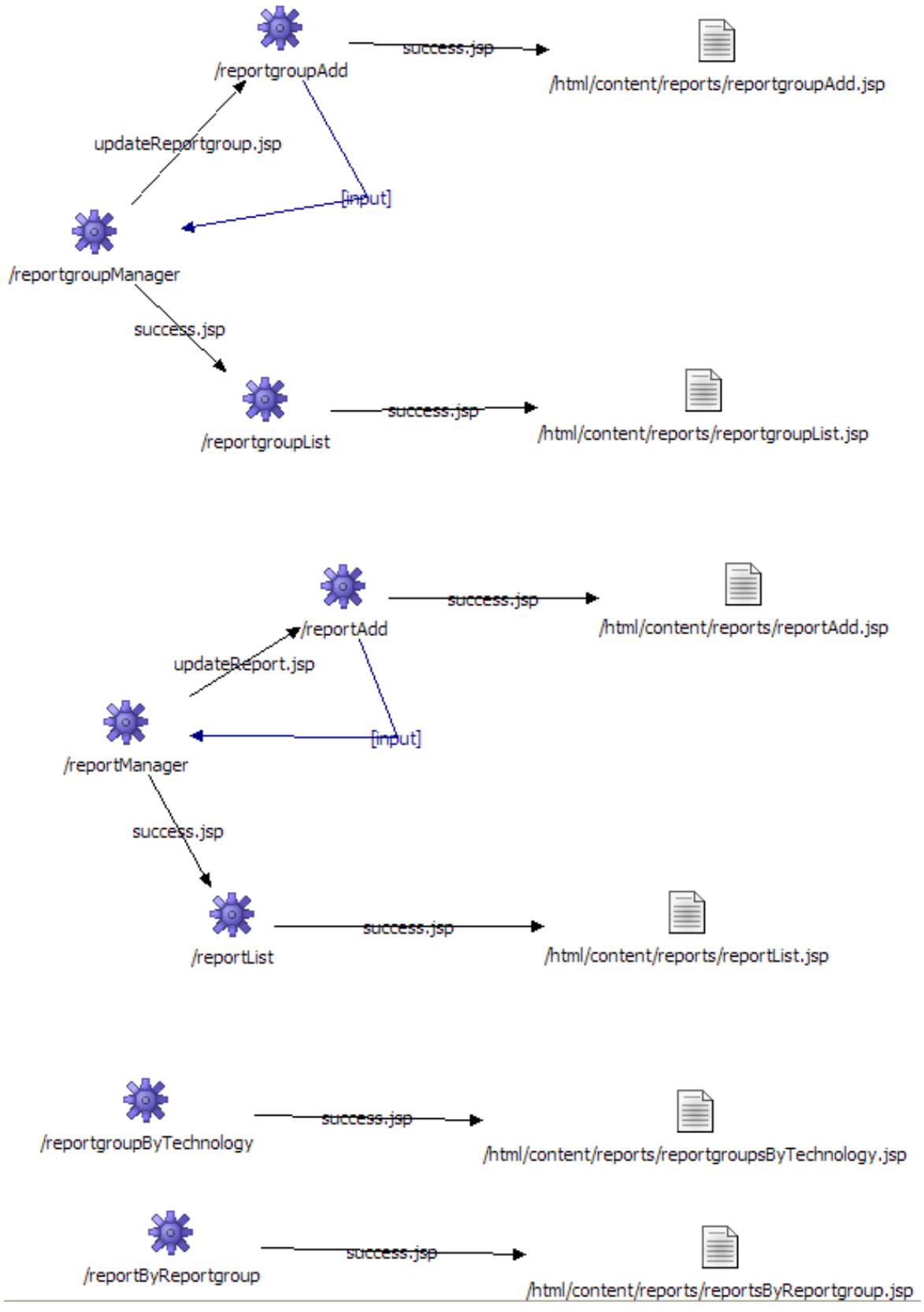
```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="com.bmw.xsapps.reporting.objects">
  <class name="ReportGroup" table="report_groups" lazy="true">
    <id name="id" type="java.lang.Long" unsaved-value="null">
      <generator class="native" />
    </id>
    <property name="name" type="string" length="255" not-null="true"
      unique-key="unique_report_groups" />
    <property name="description" type="string" length="255" />
    <many-to-one name="technology" column="technology_id"
      class="Technology" not-null="true"
      unique-key="unique_report_groups">
    </many-to-one>
    <set name="reports" inverse="true" cascade="all" lazy="true">
      <key column="report_group_id" />
      <one-to-many class="Report"/>
    </set>
  </class>
</hibernate-mapping>
```

Report.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="com.bmw.xsapps.reporting.objects">
  <class name="Report" table="reports" lazy="true">
    <id name="id" type="java.lang.Long" unsaved-value="null">
      <generator class="native" />
    </id>
    <property name="name" type="string" length="255" not-null="true"
      unique-key="unique_plants_technologies" />
    <property name="description" type="string" length="255" />
    <property name="file" column="report_file" type="string"
      length="255" unique="true" />
    <property name="hidden" column="report_hidden" type="boolean" />
    <many-to-one name="plant" column="plant_id" class="Plant"
      not-null="true" unique-key="unique_plants_technologies">
    </many-to-one>
    <many-to-one name="technology" column="technology_id"
      class="Technology" not-null="true"
      unique-key="unique_plants_technologies">
    </many-to-one>
    <many-to-one name="report_group" column="report_group_id"
      class="ReportGroup" not-null="true"
      unique-key="unique_plants_technologies">
    </many-to-one>
  </class>
</hibernate-mapping>
```

D. Struts-Konfiguration der Reporting Web-Applikation (grafische Darstellung)


success.jsp
/plantList
/html/content/reports/plantList.jsp
success.jsp
/plantManager
[input]
updatePlant.jsp
[forward]
/plantAdd
/html/content/reports/plantAdd.jsp
/technologyList
success.jsp
success.jsp
/html/content/reports/technologyList.jsp
/technologyManager
updateTechnology.jsp
[input]
success.jsp
/html/content/reports/technologyAdd.jsp
/technologyAdd
/html/content/reports/fileUpload.jsp
/html/content/reports/fileUploadSuccess.jsp
[forward]
success.jsp
/fileUpload
[input]
/fileUploadProcess


/reportgroupAdd
success.jsp
/html/content/reports/reportgroupAdd.jsp
updateReportgroup.jsp
[input]
/reportgroupManager
success.jsp
/reportgroupList
success.jsp
/html/content/reports/reportgroupList.jsp
/reportAdd
success.jsp
/html/content/reports/reportAdd.jsp
updateReport.jsp
[input]
/reportManager
success.jsp
/reportList
success.jsp
/html/content/reports/reportList.jsp
/reportgroupByTechnology
success.jsp
/html/content/reports/reportgroupsByTechnology.jsp
/reportByReportgroup
success.jsp
/html/content/reports/reportsByReportgroup.jsp

Abkürzungsverzeichnis

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
BI	Business Intelligence
BIRT	Business Intelligence and Reporting Tools
BRV	BIRT Report Viewer
CDT	C/C++ Development Tools
CE	Chart Engine
CIO	Chief Information Officer
CSS	Cascading Style Sheet
CSV	Comma Separated Value
CVS	Concurrent Version System
DAO	Data Access Objects
DE	Data Engine
DWH	Data Warehouse
EJB	Enterprise Java Beans
ETL	Extraktion Transformation Laden
GE	Generation Engine
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IPS-T	Internationales Produktions-System - Technik-Leitsystem
IT	Informations Technologie
ITIL	Information Technology Infrastructure Library
ITPM	**I**nformation **T**echnology **P**rozess Quality **M**anagement
J2EE	Java 2 Enterprise Edition
JDBC	Java Data Base Connection
JDT	Java Development Tools
JEE	Java Enterprise Edition
JSP	Java Server Pages
LDAP	Lightweight Directory Access Protocol

MIME		Multipurpose Internet Mail Extensions
MVC		Model View Controller
ODA		Open Data Access Framework
OSGi		Open Service Gateway Initiative
PDF		Portable Document Format
PE		Presentation Engine
RCP		Rich Client Platform
RCP	RD	BIRT RCP Report Designer
RD		Report Designer
RDE		Report Design Engine
RE		Report Engine
ROM		Report Object Model
SAP	BW	SAP Business Information Warehouse
SLC		Software Life Cycle
SLES		SUSE LINUX Enterprise Server
SQL		Structured Query Language
SWT		Standard Widget Toolkit
TOC		Table of Content
XML		eXtensible Markup Language

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe

- - - - - - - - - - - - - - - - - - -

Leipzig, den 29 Mai 2007 Ruslan Hrushchak